Benchtop System Users Manual

Release 4.3

Resonon Inc.

December 07, 2015

1 Introduction to Hyperspectral Imaging **3**

2 System Overview **7**

3 System Assembly **9**
3.1 Assembly of Stage, Tower, and Lighting . . . 9
3.2 Setup of Hyperspectral Imager . . . 17

4 Basic Data Acquisition **23**
4.1 Start The System . . . 23
4.2 Camera Controls . . . 25
4.3 Stage Controls . . . 25
4.4 Focus Objective Lens . . . 27
4.5 Calibrate Imager . . . 28
4.6 Scanning and Saving Datacubes . . . 30

5 Advanced Data Acquisition **33**
5.1 Camera Controls . . . 33
5.2 Stage Controls . . . 38
5.3 Additional Controls . . . 41

6 Basic Data Analysis **45**
6.1 Spectronon Tools . . . 45
6.2 Zoom, Pan, Flip, and Rotate Tool . . . 46
6.3 The Inspector Tool – Spectral Plots . . . 47
6.4 Region Of Interest (ROI) Tools . . . 48
6.5 Image Visualization . . . 53
6.6 Plot Panel . . . 57
6.7 Saving Spectra, Plots, and Images . . . 58
6.8 More Information . . . 59

7 Advanced Data Analysis 1: General **61**
7.1 File Menu . . . 61
7.2 Datacube Menu . . . 62
7.3 Image Menu . . . 71
7.4 Spectrum Menu . . . 72
7.5 Selection Menu . . . 73
7.6 Plots Menu . . . 75
7.7 Datacubes and Header Files . . . 76

8 Advanced Data Analysis 2: Hyperspectral Classification **79**

8.1 General Approach . . . 79
8.2 Spectral Angle Mapper (SAM) Classification . . . 84

9 Advanced Data Analysis 3: Hyperspectral Vegetation Indices 89
9.1 Introduction . . . 89
9.2 Generating HVI maps in Spectronon . . . 92

10 Custom Data Analysis: Writing Plugins 99
10.1 Getting started . . . 99
10.2 Spectronon Plugin Overview . . . 99
10.3 The attributes available to each plugin type . . . 101
10.4 Specs . . . 101
10.5 Using the Workbench . . . 107
10.6 Using Datacubes . . . 109
10.7 Example Plugins . . . 112

11 Batch Processing 115

12 Focusing & Calibration Sheets 119
12.1 Small Focusing Sheet . . . 119
12.2 Large Focusing Sheet . . . 121
12.3 Aspect Ratio Calibration Sheet . . . 122

13 Contact us for product support 125

14 Copyright Notice 127

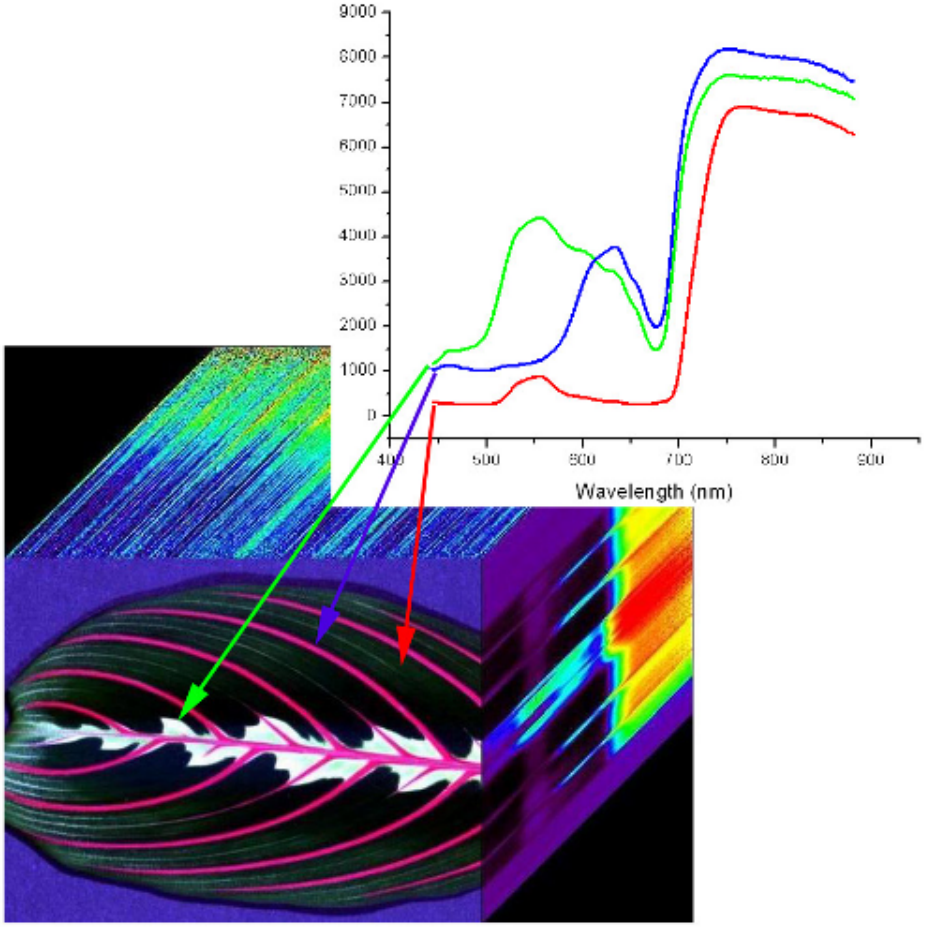
RESONON
INC

CHAPTER

ONE

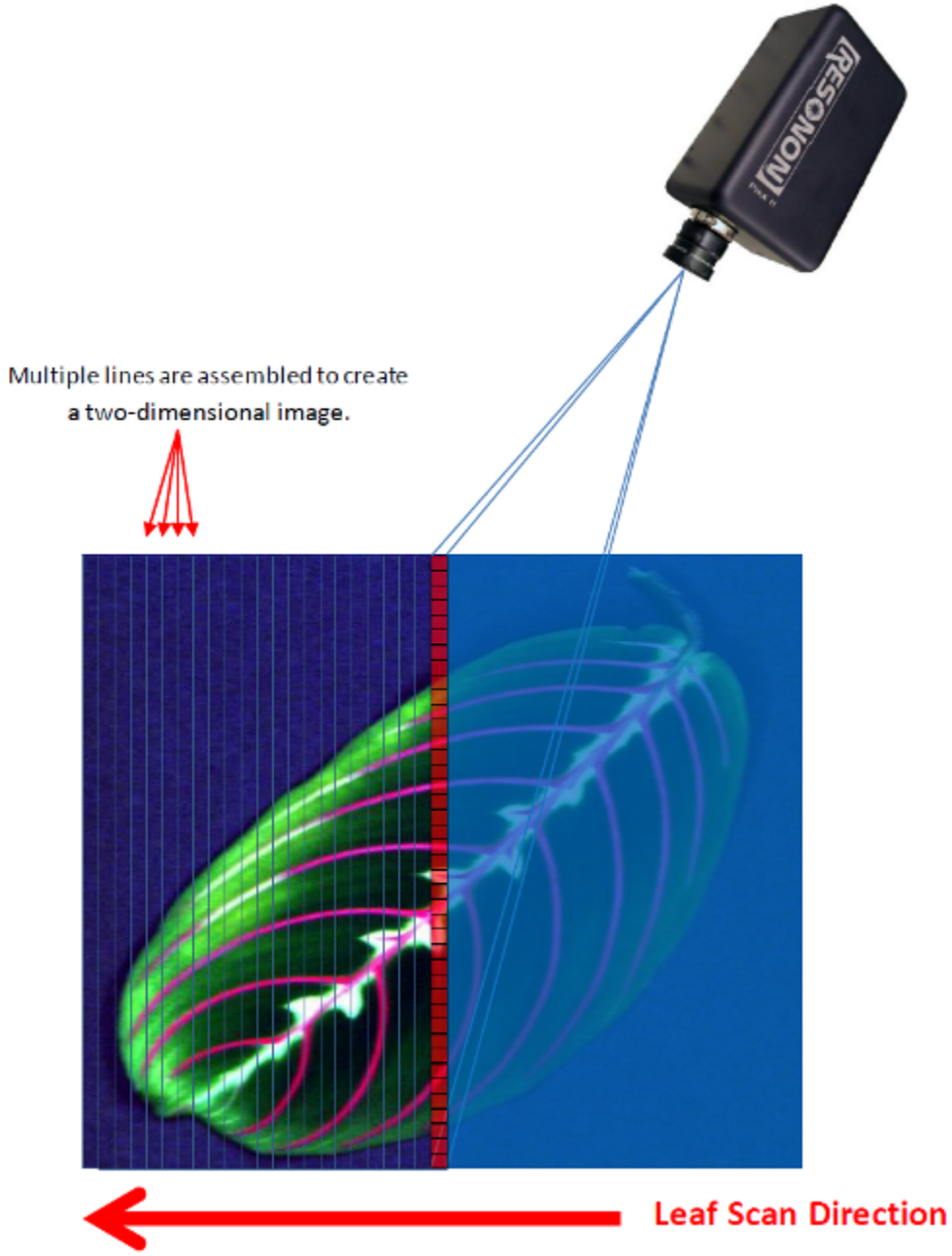
INTRODUCTION TO HYPERSPECTRAL IMAGING

Hyperspectral Imaging, or imaging spectroscopy, refers to the creation of a digital image containing very high spectral (color) resolution. Each spatial point (pixel) in a hyperspectral image represents a continuous curve of incoming light intensity versus wavelength. For example, red, green, and blue arrows below show the spectra for three pixels of an image of a leaf in the image below. The data can also be interpreted as a stack of images, with each layer in the stack representing the scene at a different wavelength – this "stack" of two-dimensional images is referred to as a "datacube."

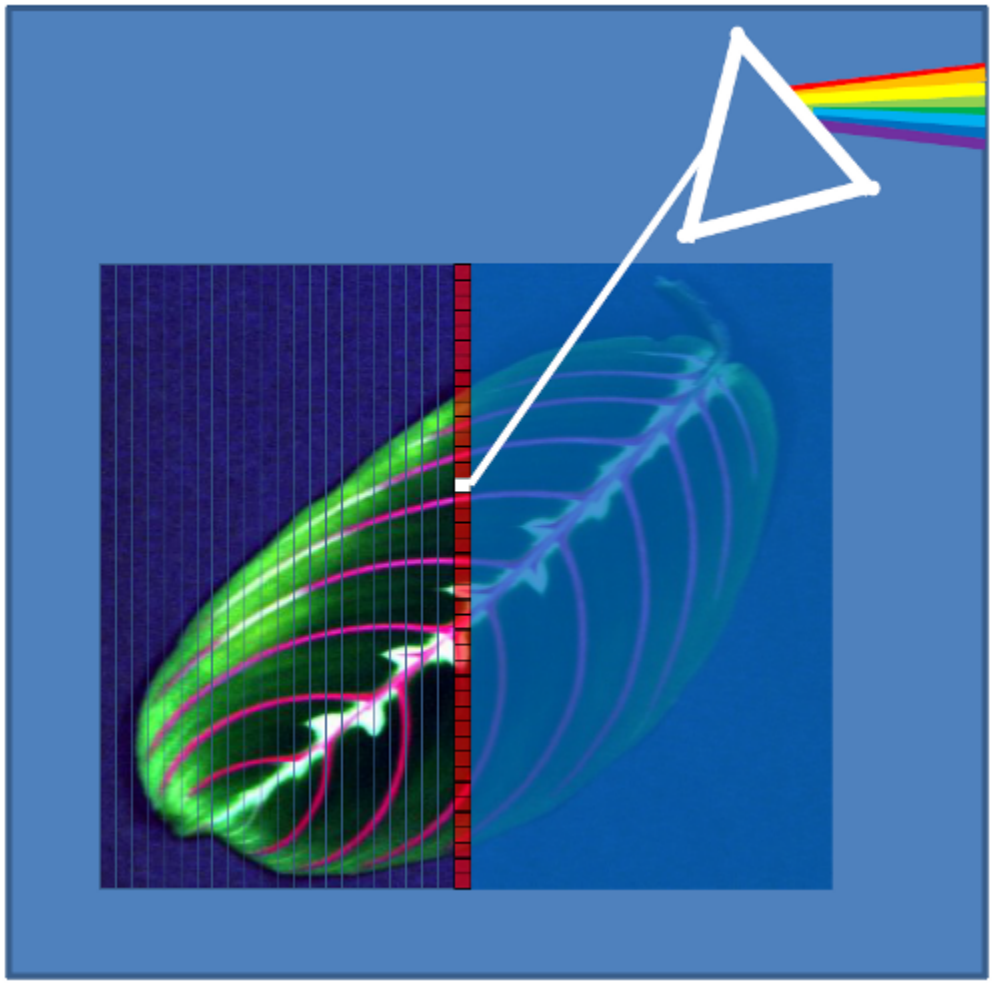


The benefit of the additional information provided by hyperspectral imaging is that it generally allows you to tell the difference between materials better than you can with traditional color images. This capability can be applied to a wide range of current and emerging applications in remote sensing, sorting, quality control, research and development, and more.

Resonon imaging spectrometers are **line-scan** imagers, which means they collect data one line at a time. To assemble a complete two-dimensional image, multiple lines are imaged as the object (or imager) is translated. The multiple line-images are then assembled line-by-line to form a complete image, as indicated below. A scanning system is often needed to use a Resonon imaging spectrometer.



To obtain hyperspectral data, the signal from each pixel is dispersed (or diffracted) into its spectral components, much like passing the light from each pixel through a prism. This process occurs for every pixel in the line (red squares in the image below). The dispersed signals from all the pixels are imaged onto a focal plane array. One of the benefits of this approach is that all the spectral (color) information is collected at the same time for each pixel. The result is a detailed spectral curve for every pixel in the image.



CHAPTER

TWO

SYSTEM OVERVIEW

Resonon's benchtop hyperspectral imaging system is comprised of a Pika hyperspectral imaging camera, linear translation stage, mounting tower, lighting assembly, and software control system. The positions of the imager and lighting assembly are adjustable along the length of the tower. See Figure 1 below.

Resonon's hyperspectral imagers are line-scan imagers (also referred to as push-broom imagers). Two-dimensional images are constructed by translating the sample relative to the camera. This is typically accomplished by placing the sample on a linear translation stage.

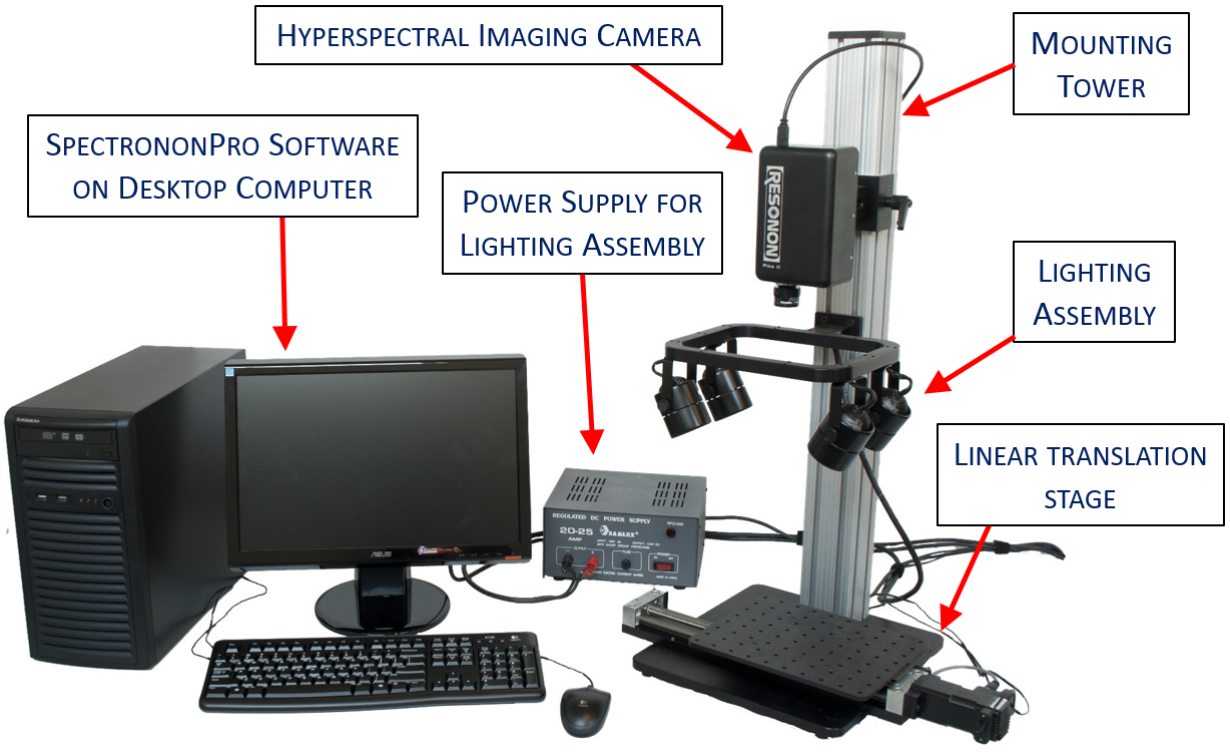
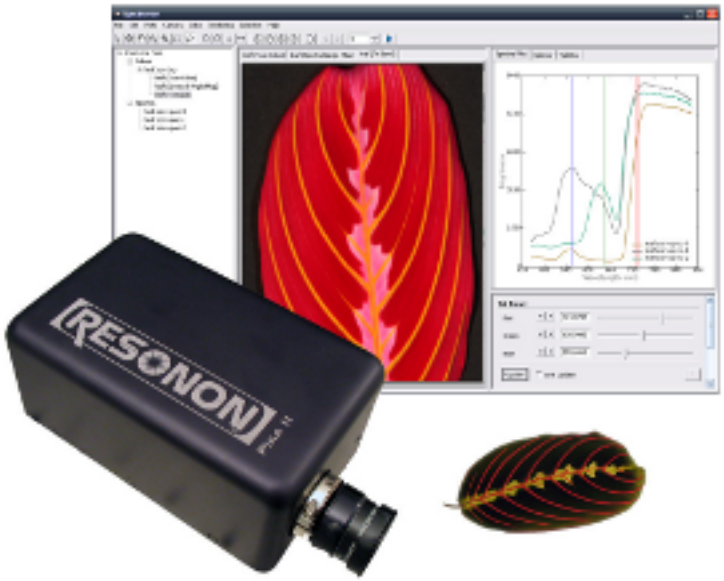


Fig. 2.1: Benchtop hyperspectral imaging system

Resonon's Pika imaging spectrometers are compact, high fidelity, digital instruments for industrial and scientific ap-

plications. Spectronon is a powerful hyperspectral data visualization and analysis software package we provide as a free download. Spectronon is easy to learn, offers efficient workflow, and is highly extensible by the user for custom applications. Additionally, a number of datacubes can be downloaded from the Resonon website (www.resonon.com) so you can begin exploring hyperspectral data within a few minutes.



SpectrononPro has all the features of Spectronon, but also includes data collection tools that are highly integrated with our Pika imaging spectrometers to streamline the collection of spectral images.

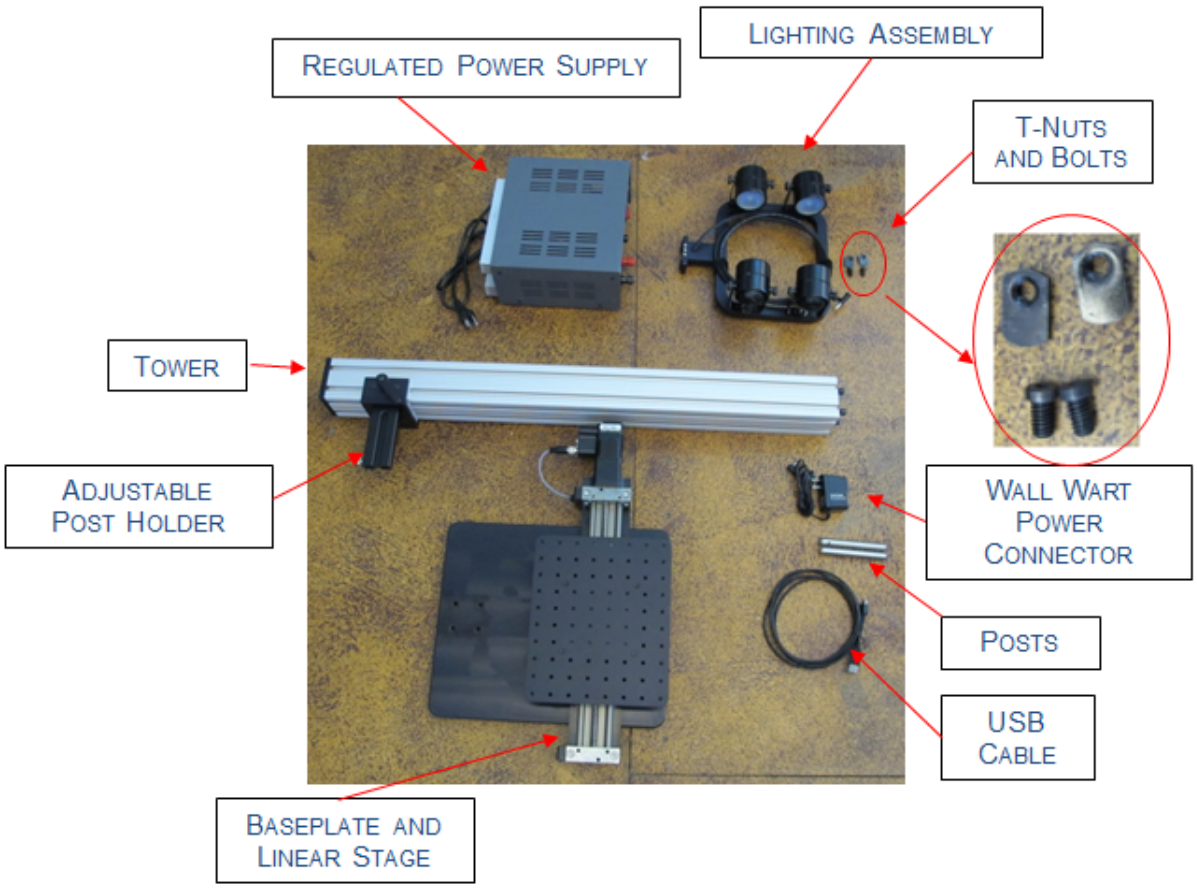
CHAPTER

THREE

SYSTEM ASSEMBLY

3.1 Assembly of Stage, Tower, and Lighting

Begin system assembly by removing all items from the shipping package. Confirm that the items shown in the images below. If you are missing components, please contact Resonon.



Install lighting onto the tower. Begin by sliding the T-nuts into the slots on the tower. Slide in one T-nut into each slot on the same side as the adjustable post holder.

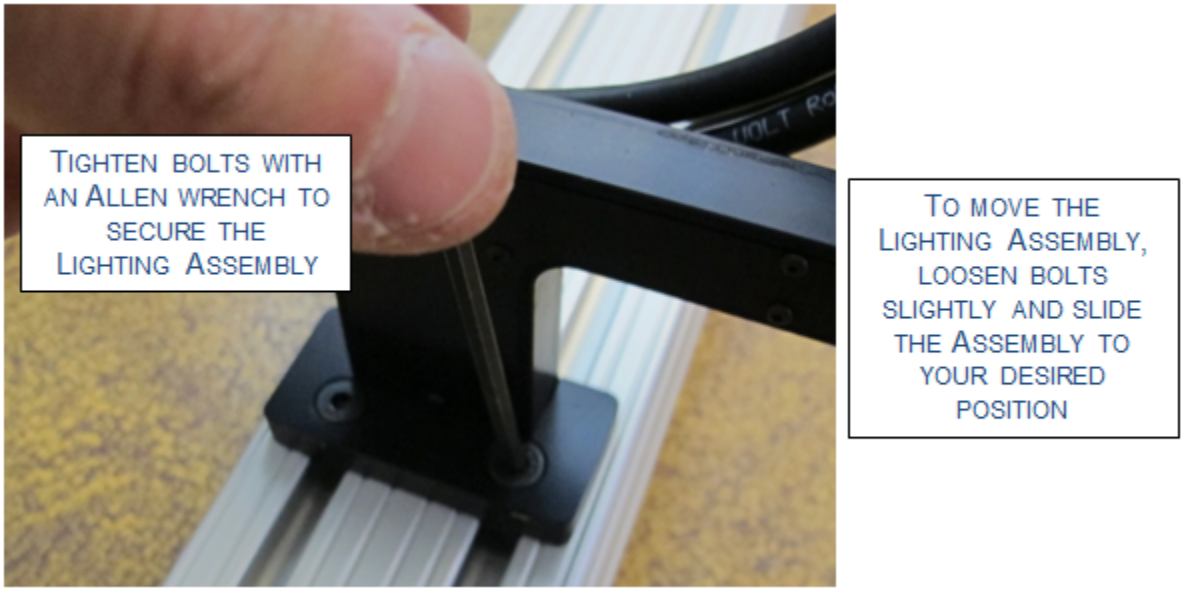
SLIDE T-NUTS INTO THE GROOVES OF THE TOWER

Position the T-nuts side-by-side at the location you wish to attach your lights. (Note: You may easily adjust the lighting position later.)

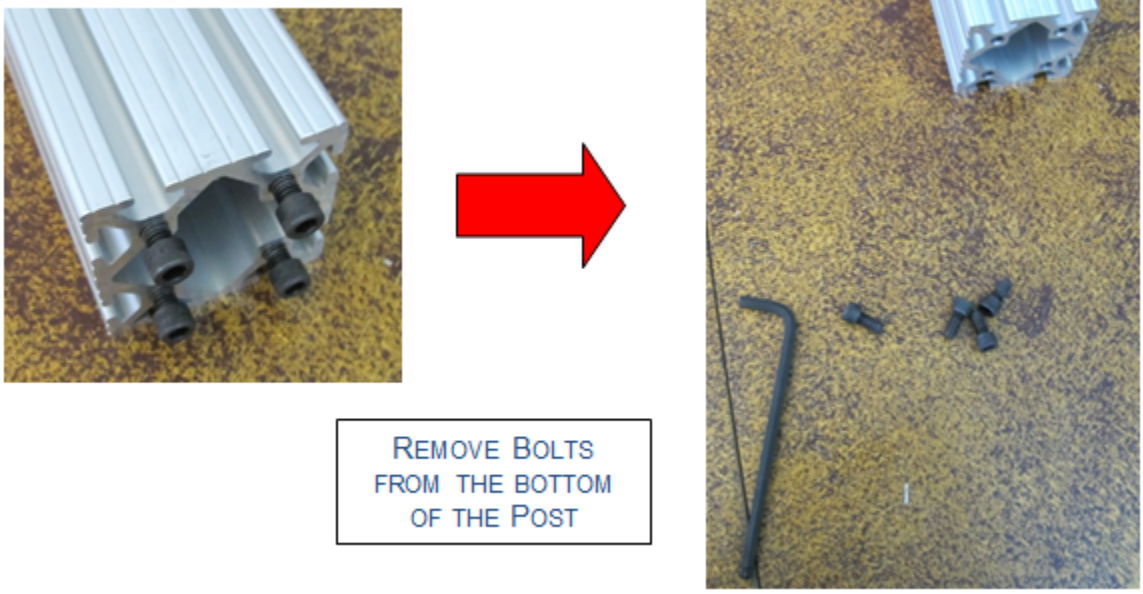
POSITION T-NUTS SIDE-BY-SIDE AT THE DESIRED LOCATION FOR YOUR LIGHTS

Place the lighting assembly on the post such that the mounting holes line up with the T-nuts.

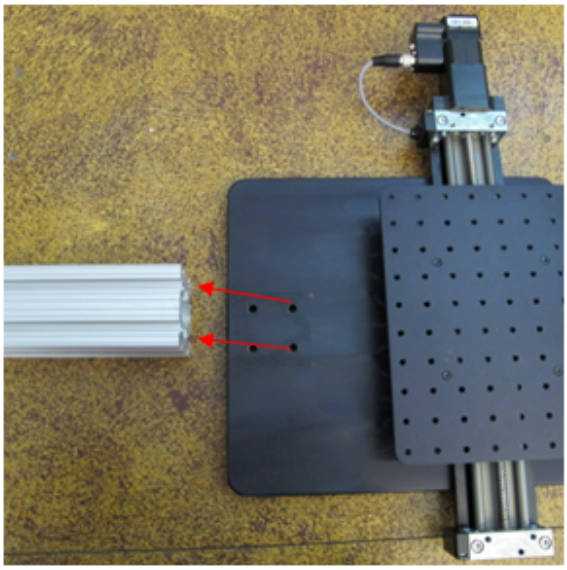
Attach the lighting assembly to the T-nuts using the supplied bolts.


TIGHTEN BOLTS WITH AN ALLEN WRENCH TO SECURE THE LIGHTING ASSEMBLY
TO MOVE THE LIGHTING ASSEMBLY, LOOSEN BOLTS SLIGHTLY AND SLIDE THE ASSEMBLY TO YOUR DESIRED POSITION

Next, attach the post to the baseplate. Begin by removing the four bolts from the bottom of the post.



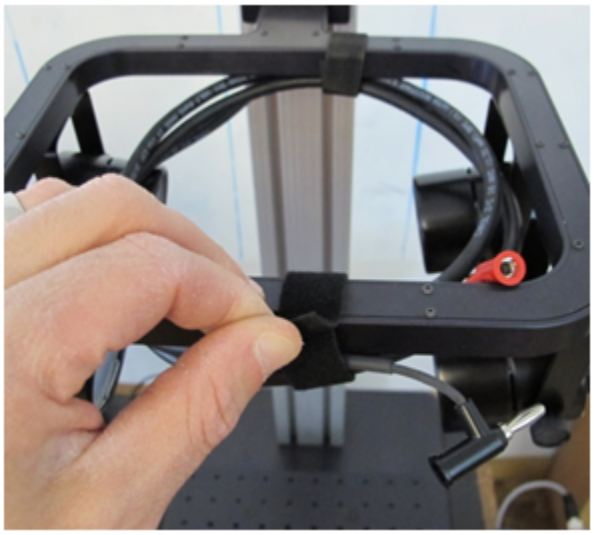
Align the post to the baseplate, then tip the baseplate up and attach using the four bolts just removed from the bottom of the post.



After tightening all four bolts, tip the tower upright.

Connect the lighting assembly to the regulated power supply.

REMOVE THE HOOK-AND-LOOP TABS HOLDING THE WIRING TO THE LIGHTING ASSEMBLY.

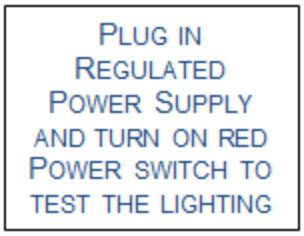


Take the red and black banana plugs, and insert them into the red and black connectors on the regulated power supply.

Push the wire into a slot on the side of the tower.

PUSHING THE WIRE INTO A TOWER SLOT PROTECTS IT FROM THE LAMP HEAT

Plug the regulated power supply into a 120 V socket and turn on to test the lighting. Turn the lights off after testing to continue with assembly.

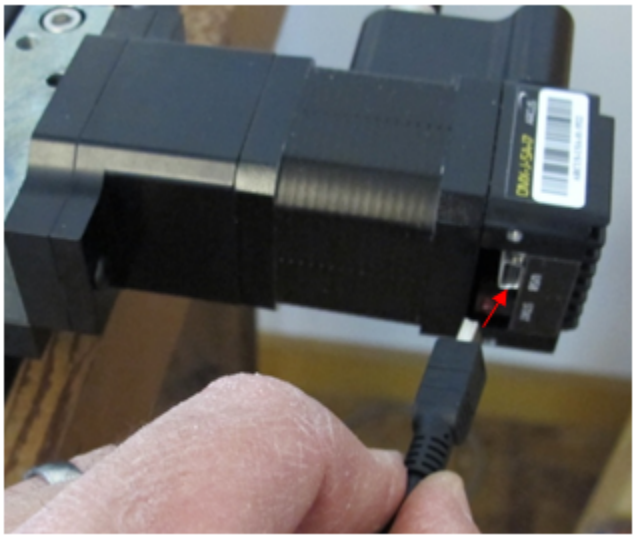


If the lights do not turn on, turn off the red power switch and check all connections. If the lights continue to fail, contact Resonon.

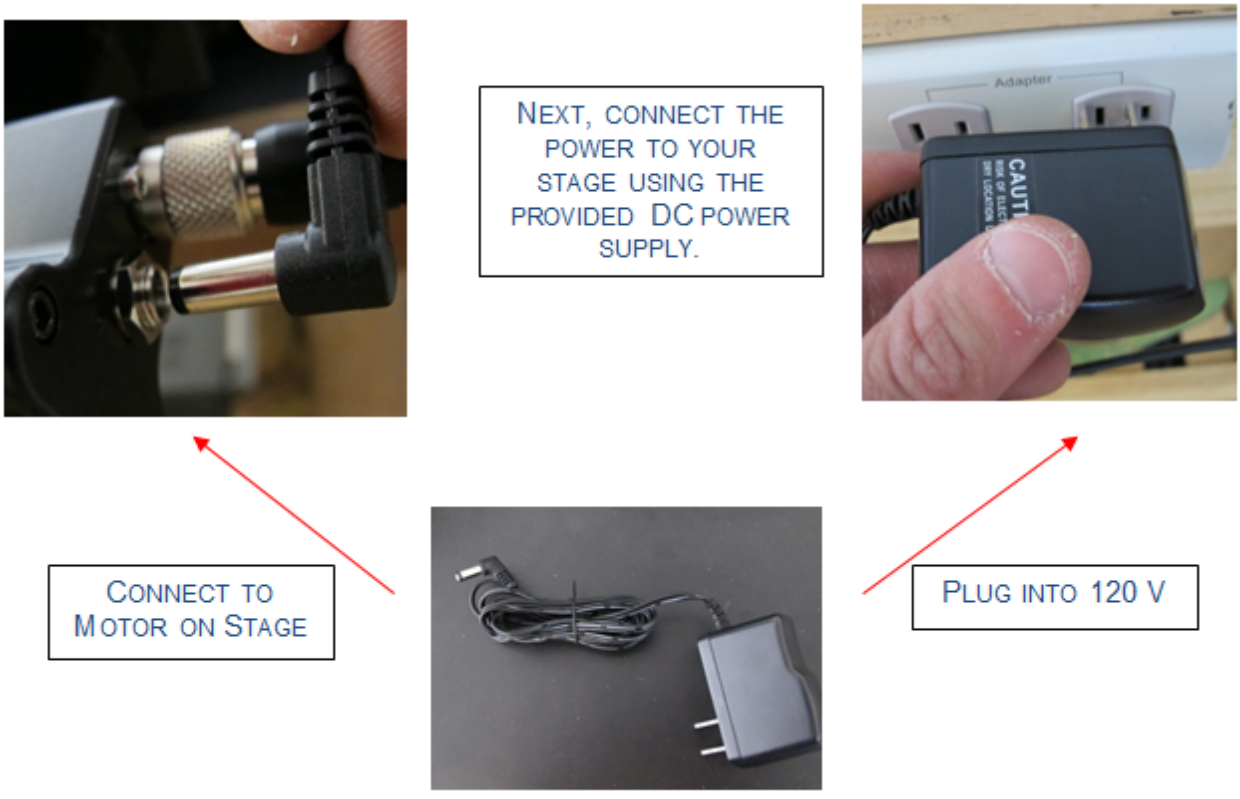
Note: When illuminating your sample with Resonon's lighting system, allow 20 minutes for the lighting to fully stabilize.

Using the supplied USB cable, connect the stage to your computer. Plug the cable into a black USB 2.0 port, not a blue USB 3.0 port.

Plug the mini-USB connector end into the stage motor.

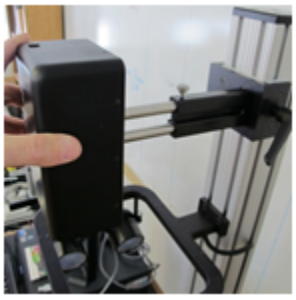


Next, connect the power to the stage using the provided DC power supply.



Screw the posts onto the bottom of your Pika imaging spectrometer. The studs in the posts are ¼"-20. Install both posts in neighboring mounting holes on the bottom of the imaging spectrometer.

Insert the posts into the adjustable post holder on the tower.



Tighten setscrew on the adjustable post holder to secure your Pika imaging spectrometer.

To adjust the height of your Pika imaging spectrometer, loosen the handle on the side of the adjustable post holder, move to your desired position, and then re-tighten the handle.

Assembly of your linear scanning stage system with stabilized lighting assembly is now complete.

3.2 Setup of Hyperspectral Imager

3.2.1 Connecting Pika imaging spectrometer

All Pika imaging spectrometers must be interfaced to a computer for data acquisition. Depending on which Pika imaging spectrometer you have, this interface may utilize a USB, Ethernet, Firewire 800 (IEEE 1394b), or CameraLink interface. Additionally, some Pika spectrometers require external power. The images below show interfacing with Firewire 800.

Using the provided interface cable, connect your Pika imaging spectrometer to the appropriate port on your computer.

When using a Pika II camera with a GigE interface, the camera should be plugged directly into the computer's network adapter (not connected to the computer via a local area network, hub, or switch).

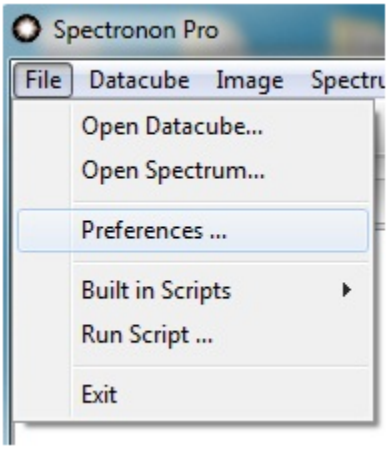
3.2.2 Installing Calibration Values

During initial setup you will input the calibration numbers for your Pika imaging spectrometer into SpectrononPro software. The calibration numbers provide the information needed for SpectrononPro to provide the correct wavelength values for the measured spectral bands. Your Pika's calibration numbers are provided on a sheet that comes with your imaging spectrometer. You only need to do this during initial setup, or after an instrument re-calibration. If you cannot locate your calibration numbers, contact Resonon at: ProductSupport@resonon.com.

Connect your Pika imaging spectrometer to your computer. Launch SpectrononPro by double-clicking on the SpectrononPro icon

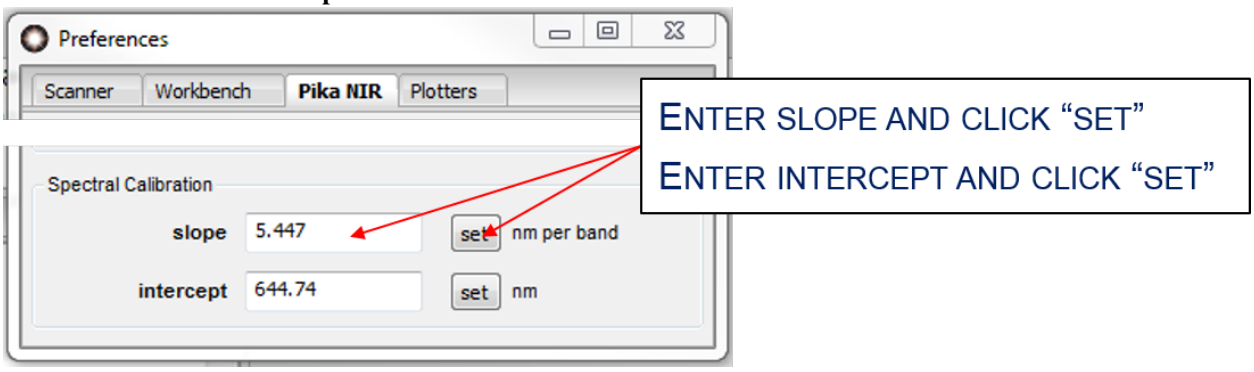
, or navigate to it from your Start menu.

From the Main menu, select **File** → **Preferences...** This will reveal the **Preferences** menu. Select the tab for your Pika imaging spectrometer.



Locate the value for **Slope** on the calibration sheet provided with your Pika imaging spectrometer. Enter this value in the window labeled **slope** in the **Preferences** menu and then click on **set**.

Locate the value for **Intercept** on the calibration sheet provided with your Pika imaging spectrometer. Enter this value in the window labeled **intercept** in the **Preferences** menu and then click on **set**.

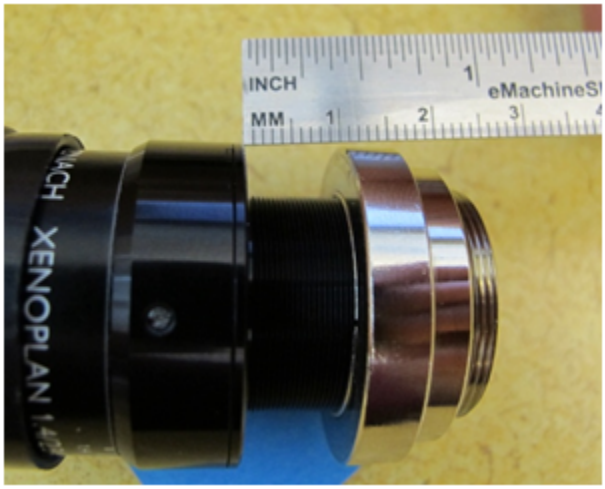


Initial setup is now complete. Instructions for how to collect hyperspectral images are given in the chapters on data acquisition.

3.2.3 Objective Lens Installation

Resonon imaging spectrometers are designed to operate with Schneider c-mount lenses. To install the Schneider lens, first remove the lens and imager from their packaging. Unlock the set screw on the chrome lens collar of the Schneider lens with a 2 mm hex wrench. NOTE: Do NOT remove the screw all the way, just loosen it a few turns.

Holding onto the objective lens by the chrome collar, unscrew the black lens barrel out until there is approximately a 10mm gap between the barrel and collar.



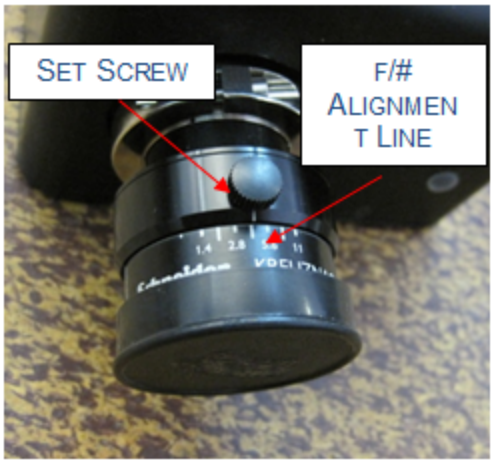
Do **NOT** install the lens onto the imager if the barrel is screwed in too far as shown below.

While holding and squeezing the chrome collar, thread the objective lens onto the imager.

The objective lens is now installed.

3.2.4 Setting Objective Lens Aperture

Next, **set the f-stop (f/#)** of the objective lens. There is a range of acceptable options and the end user will need to choose the appropriate value based on his or her application. Large f/#s will provide large depths of field, but limit the amount of light collected, whereas small f/#s will have small depths of field but will exhibit better light gathering.



To set the f/# on Schneider lenses, for example, loosen the setscrew on the objective lens itself. This will allow you to rotate the f/# collar on the objective lens, which can be read by noting the location of a small white alignment line. Align the white line with your desired f/#, then retighten the setscrew.

Acceptable f/#s

Pika II	3 and larger
Pika XC	2.4 and larger
Pika NIR	1.8 and larger

3.2.5 Installing SpectrononPro Software (if necessary)

Your hyperspectral imaging system will be delivered with SpectrononPro already installed. If however you need to reinstall the software, it will be necessary to set the appropriate version flags. To do so, create an executable shortcut on the desktop, and right click on the shortcut to bring up the **Properties** window. On the **Shortcut** tab, in the *Target* field, add the following flag relevant for your imager to the executable name:

Imager	Flag
Pika II	none
Pika XC	-calli
Pika NIR	-cxene

CHAPTER

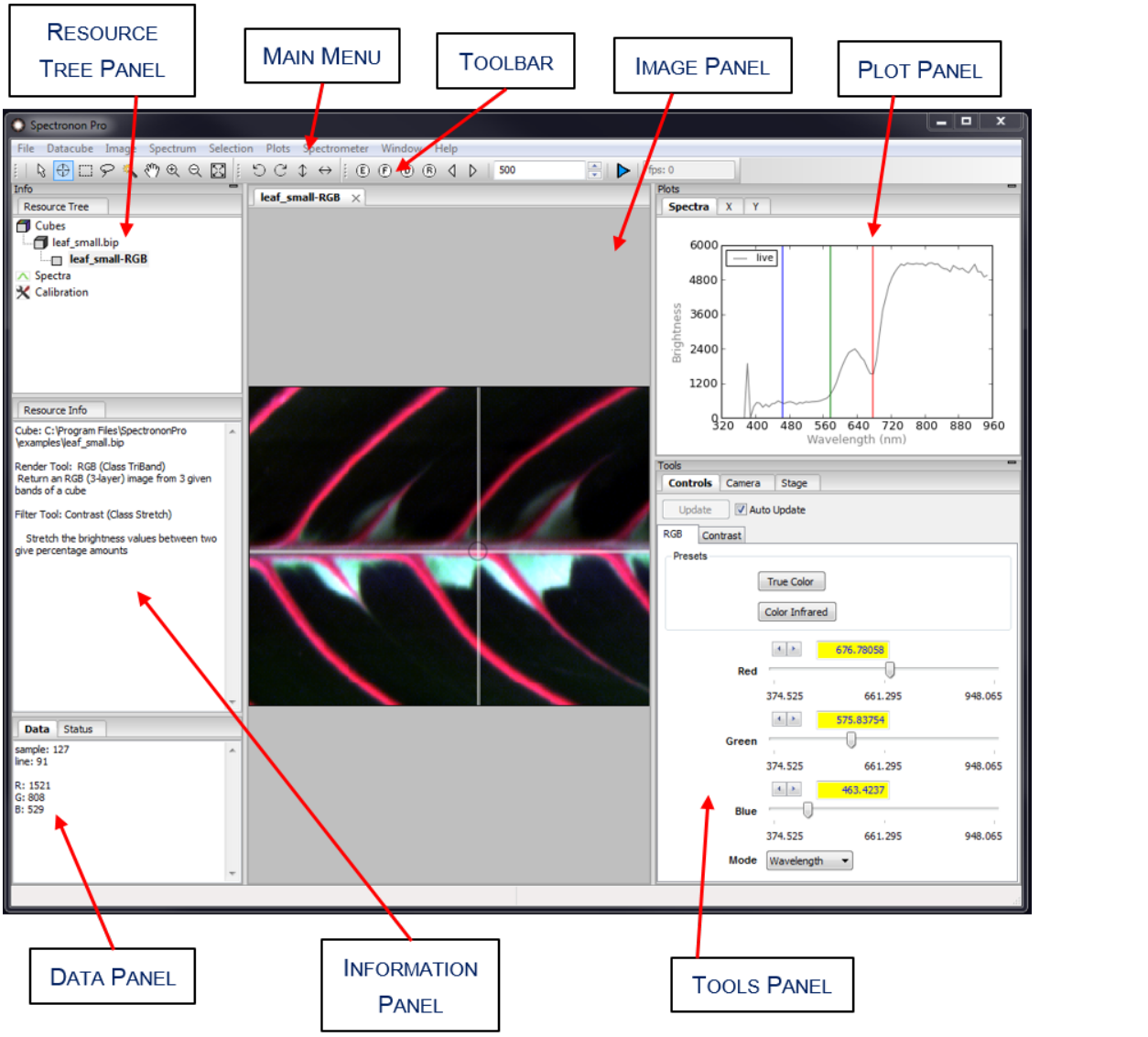
FOUR

BASIC DATA ACQUISITION

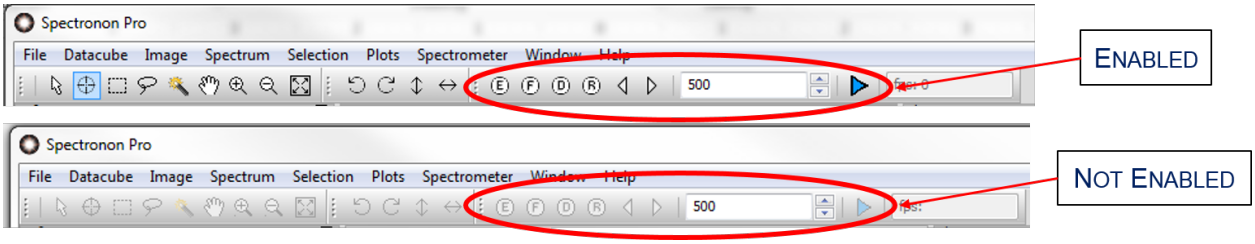
4.1 Start The System

If you have a lighting system, turn on the lights and let them warm up. It may require 15 to 20 minutes for the illumination to fully stabilize.

With the camera and scanning system connected to your computer, launch **SpectrononPro** software by double-clicking on the SpectrononPro icon or starting SpectrononPro from your Start menu. The SpectrononPro user interface is shown below with the various windows labeled.



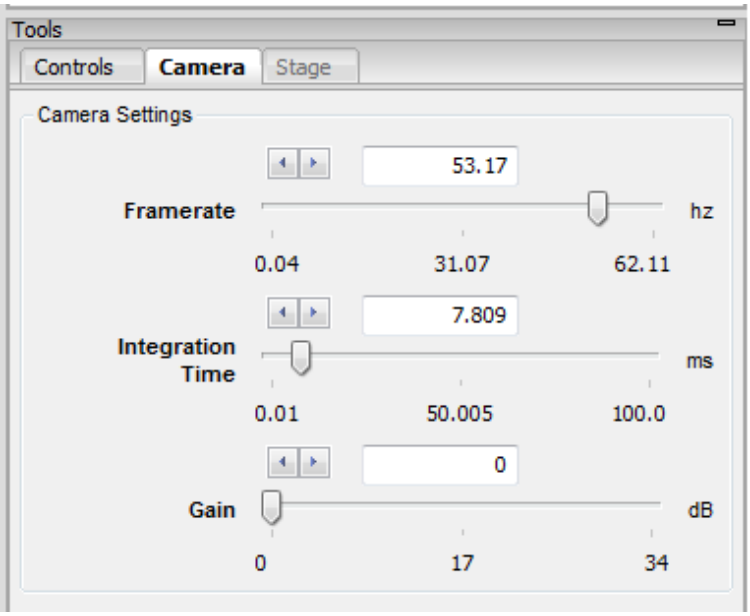
Once the software has started, make sure that the imager and stage controls (if used) are enabled. The imager and stage tools will be greyed out if not enabled, as shown below.



You can get the latest version of SpectrononPro by clicking on **Help** → **Check For Updates**. This won't download the latest version, but will give an alert if there is a newer version.

4.2 Camera Controls

Exposure parameters can be controlled by clicking on the **Camera** tab in the **Tools Panel**.



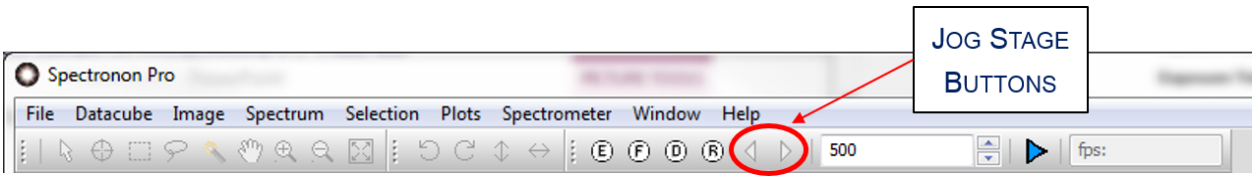
Frame Rate is equal to the number of images acquired each second, and limits the maximum exposure time (Max Exposure Time = 1.0 ÷ Frame Rate).

Integration Time (also known as **Exposure Time**) is the duration of data acquisition for each individual line image.

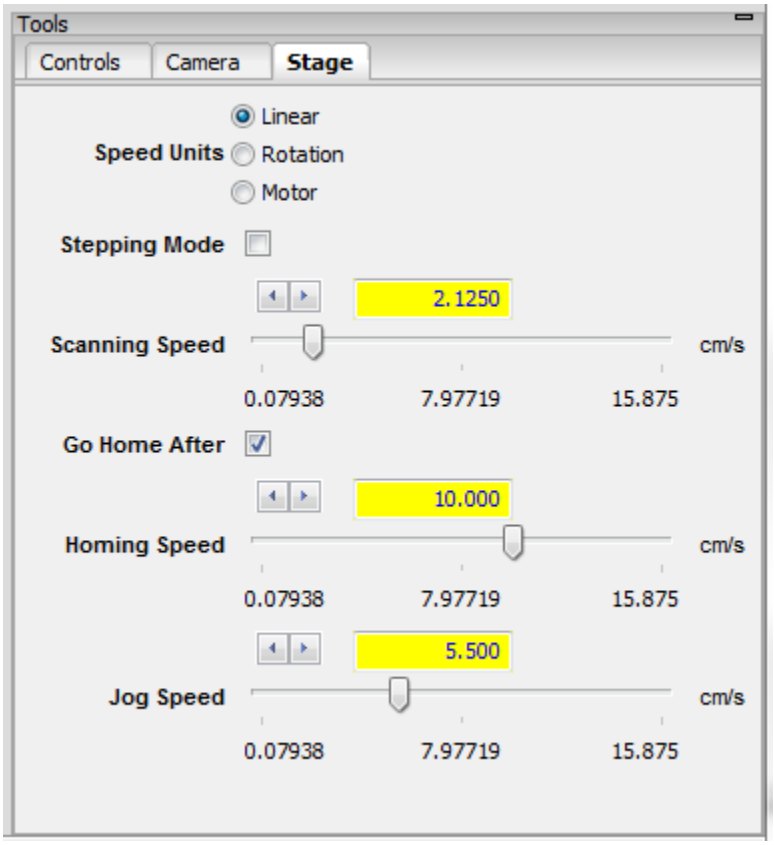
Gain is a factor which increases the signal, but at the expense of signal-to-noise ratio. Try to keep gain as low as possible (preferrably zero), unless absolutely necessary. (Note: The Pika NIR camera does not have a gain control tool.)

4.3 Stage Controls

You can move the stage manually by clicking on the **Jog Stage** buttons, located on the tool bar of SpectrononPro. The buttons will move the stage incrementally in either direction. Use the buttons to center the stage underneath the Pika imaging spectrometer.



The stage can be further controlled by clicking on the **Stage** tab in the **Tools Panel**.



Speed Units is a setting used for different types of stages. **Linear** is used for the standard linear translation stage that is installed on most benchtop systems. **Rotation** is used for a tripod-mounted rotational scanning stage, typically used in outdoor applications. **Motor** displays speed in "motor pulses per second."

Stepping Mode controls the way the stage moves in relation to the imager. When the Stepping Mode box is not checked, the stage and imager both run continuously during the scan time.

When the Stepping Mode box is checked, the stage moves incrementally, an image is acquired while the stage is stationary, the stage moves incrementally again, another image is acquired while the stage is stationary, and so on. This behavior is preferrable when the scanning speed is very slow, when the integration time is very long, or to guarantee there is no motion blur in your scan.

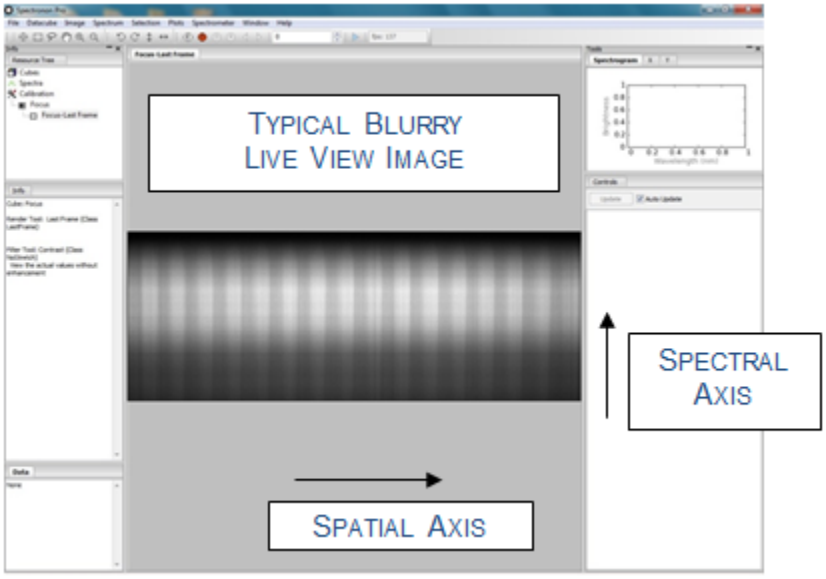
Scanning Speed is the linear speed of the stage during a scan.

If the "Go Home After" box is checked the stage will return to its starting position after a scan. The speed at which the stage returns to its original position is the **Homing Speed**, and the **Jog Speed** is used for the **Jog Stage** buttons, described above.

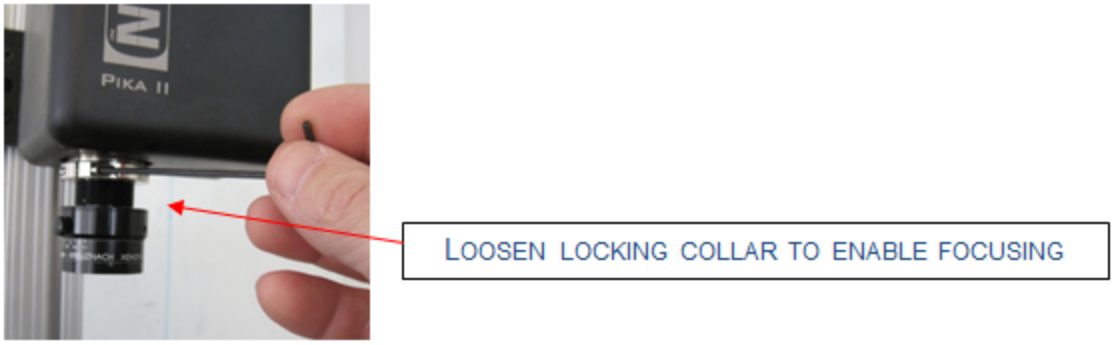
4.4 Focus Objective Lens

You are now ready to **focus the objective lens** of your Pika imaging spectrometer. At first, this process is somewhat challenging, but with a little practice it becomes quite easy. Begin by clicking on the **Focus** button Ⓕ located on the SpectrononPro tool bar. This will reveal a live image from the camera within your Pika imaging spectrometer. (Wave your hand in the field of view of your Pika imaging spectrometer to confirm that the image is a live view.) One axis of this image represents the spatial (position) axis of your object, and the other is the spectral (wavelength) axis. (To understand this view better, move colored objects within the field of view of your imager after you have focused the objective lens.)

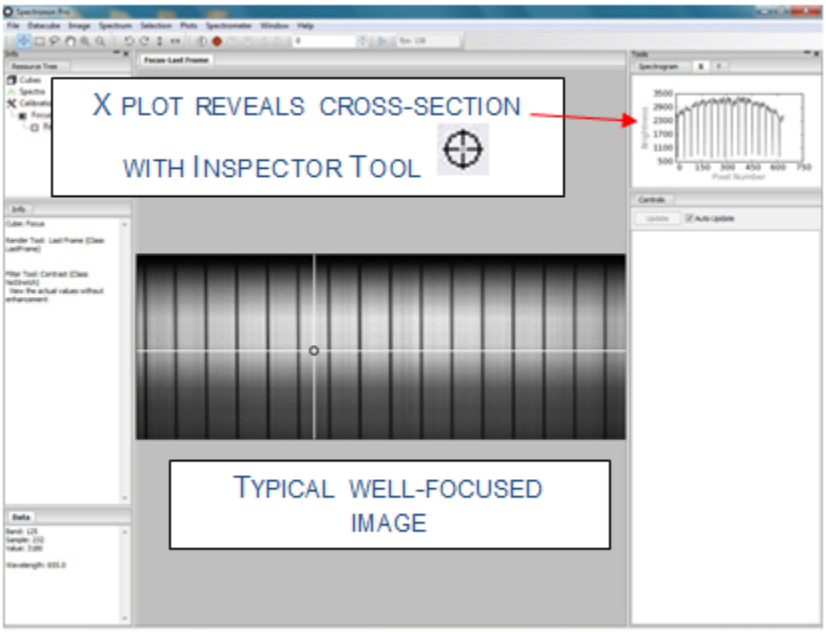
Place an object with multiple light and dark regions within your Pika imaging spectrometer's field of view. A sheet of paper with dark lines, as provided in the chapter titled **Focusing & Calibration Sheets**, works well. If you are in the field and are observing objects at a distance, direct your Pika towards an object with multiple features, such as a tree with many branches. Unless your lens is already focused, you will see a series of blurry or barely discernable lines in the Image Panel of SpectrononPro.



To adjust the focus, first unlock the focus adjustment. With Schneider lenses, this is done by loosening the locking metal collar on your objective lens using an Allen wrench, size 5/64 inch.



Then rotate the objective lens until you see dark lines from your object come into focus, as shown. Maximize the sharpness of the lines.



Hint: Clicking the Inspector Tool in the Image Panel, and then selecting X tab in the Plots window will reveal a cross-section plot of your image. Viewing this plot allows you to graphically see the sharpness of your focusing. You can zoom in on the X-axis by clicking the Zoom tool and then clicking on the X-axis.

Once you have completed focusing, re-tighten the lock to the focus adjustment. Then click on the Focus tool again to toggle the camera live view off.

Hint: See our You Tube video on focusing at http://www.youtube.com/resonon.

4.5 Calibrate Imager

The following discussion describes how to set up your system to scan for reflectance scaled to a reference panel. More advanced options are described in the Appendix.

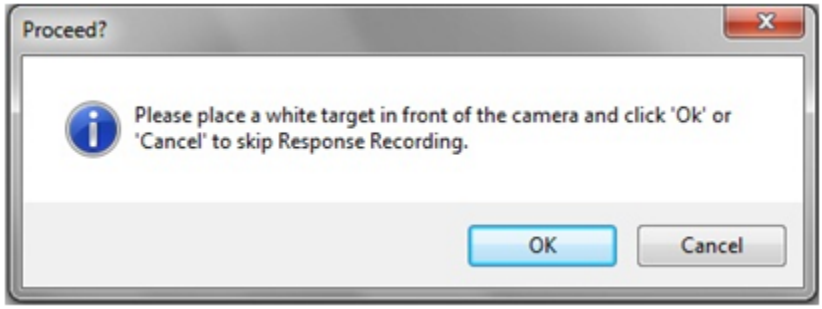
4.5.1 Remove dark current

SpectrononPro makes it easy to remove the average dark current noise from your scans. Begin by clicking on the **Dark Current** button Ⓓ on the SpectrononPro toolbar. You will be instructed to block all light entering your Pika imaging spectrometer by blocking the objective lens.

Once you have the objective lens blocked, click **OK** as instructed. SpectrononPro will then collect multiple dark frames and use these measurements to subtract the dark current noise from your measurements. The **Dark Current** button on the toolbar will appear with a red check through it as soon as the dark frames have been collected. Once you see the red check, unblock the objective lens.

4.5.2 Set Reflectance Reference

Measuring absolute reflectance of an object requires calibration to account for illumination effects. To do this, click on the **Response Correction Cube** button Ⓡ on the SpectrononPro toolbar. A message will appear telling you to place a reference material within your Pika imaging spectrometer's field of view. The reference material should be uniform across the imager's field of view. Examples of reference materials include Spectralon® or sheets of white Teflon.



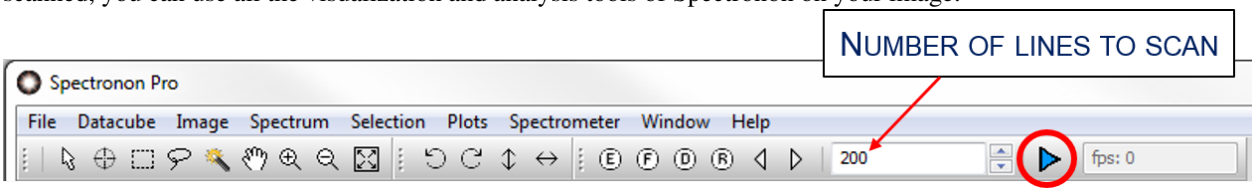
Once the reference material is in place, click on **OK**. This will trigger a short scan of the reference material. Once complete, the **Response Correction Cube** will appear with a red check mark, indicating that the data you collect will be scaled in reflectance to your reference material, including flat-fielding to compensate for spatial variations in your lighting.

Note: The **Dark Current** button and the **Response Correction Cube** button are disabled in live camera view mode. Click on the **Focus** Ⓕ tool to toggle the camera live view off.

Once the imager is calibrated for both dark current and reflectance reference, the imager will remain calibrated until the references are removed by the user, or the machine is turned off. If the integration time is changed by the user after calibration, the reference signals will be adjusted accordingly. To manually remove the references, click **Spectrometer → Remove Dark Current Cube**, and **Spectrometer → Remove Response Correction Cube**.

4.5.3 Adjust Aspect Ratio

To scan an image, type in the number of lines you would like to scan in the window just to the left of the **Scan Button** . A number such as 200 is often good initially. Then click on the scan button. A waterfall image will appear in the Image Panel of SpectrononPro. Increase or decrease the number of lines to scan as desired. Once your image is scanned, you can use all the visualization and analysis tools of Spectronon on your image.



Note: You can stop a scan by re-clicking on the Scan Button .

Your first image may appear distorted in one dimension. To correct this distortion, you must adjust the step-size of your scanning system. (Recall that Pika imaging spectrometers are line-scan instruments. Thus, by adjusting the step-size of the scanning system, you are adjusting the spacing of the lines used to assemble your image.)

To adjust your image's aspect ratio, it is useful to image an object whose distortion is easy to observe, such as a circle. For example, print out **Pixel Aspect Ratio Calibration Sheet** provided in the **Focusing & Calibration Sheets** chapter of this manual.

Place an object with circles within the field of view of your Pika imaging spectrometer and record a scan with enough lines that you can see the complete circle. You may need to record several trial images to determine how many lines to scan.

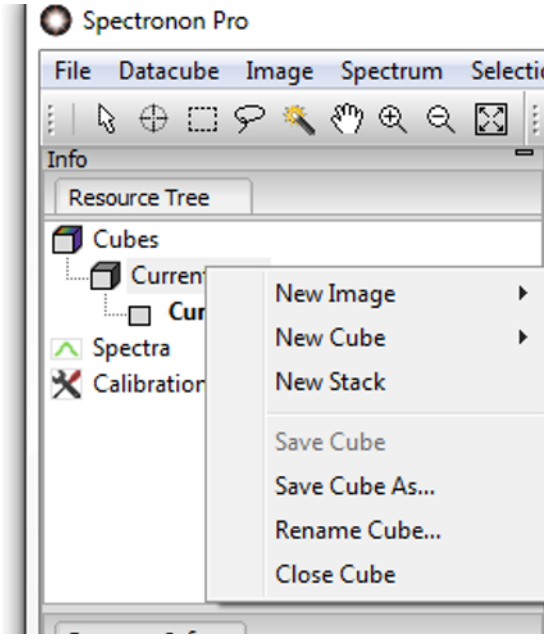
If your image is distorted along the scan direction, change the **scanning speed** on the **Stage** tab on the **Tools** panel. If your image is elongated along the scan direction, decrease the scanning speed; conversely if your image is too short along the scan direction, increase the scanning speed.

After resetting the steps per scan, record a new image to observe the change in image distortion. Repeat the above process until your image is no longer distorted.

4.6 Scanning and Saving Datacubes

To record a hyperspectral datacube (image), put the number of lines you wish to scan in the window to the left of the Scan button Then press the Scan button to record a datacube.

A waterfall image of your datacube will appear in the Image Panel of Spectronon, and a new entry labeled **Current Scan** will appear in the **Resource Tree**. To save the scanned datacube (image), use your mouse to select **Current Scan** and then either right-click or select **Datacube** → **Save Cube**. This will open a new window that allows you to name the datacube and save it in a folder of your choosing.



If you do not save your datacube, the **Current Scan** will be overwritten when you record another datacube. A warning will appear.

CHAPTER

FIVE

ADVANCED DATA ACQUISITION

5.1 Camera Controls

Control of the imager can be controlled from two windows: the **Camera** tab in the **Tools Panel**, and the **Preferences** window (accessed by **File** → **Preferences**).

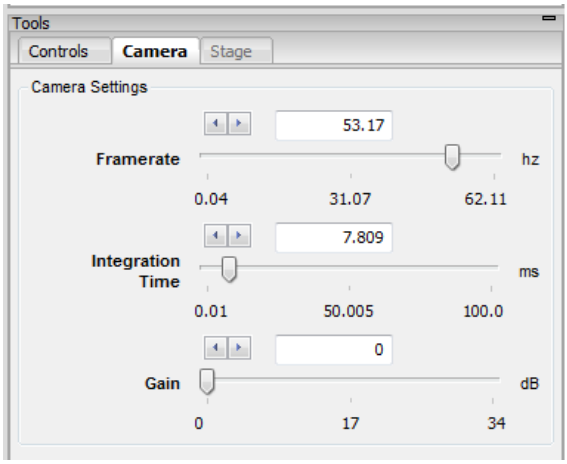
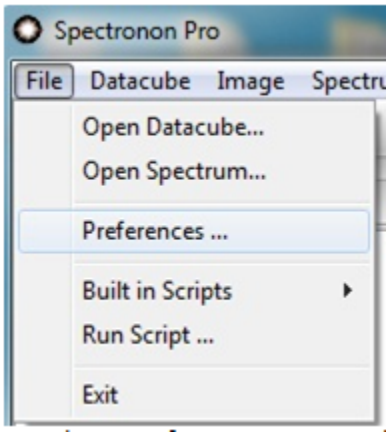


Fig. 5.1: **Camera tools in Preferences window**

When the Preferences window opens, select the tab for your specific Pika imager (Pika II, Pika XC, or Pika NIR). The options for different Pika models will vary.



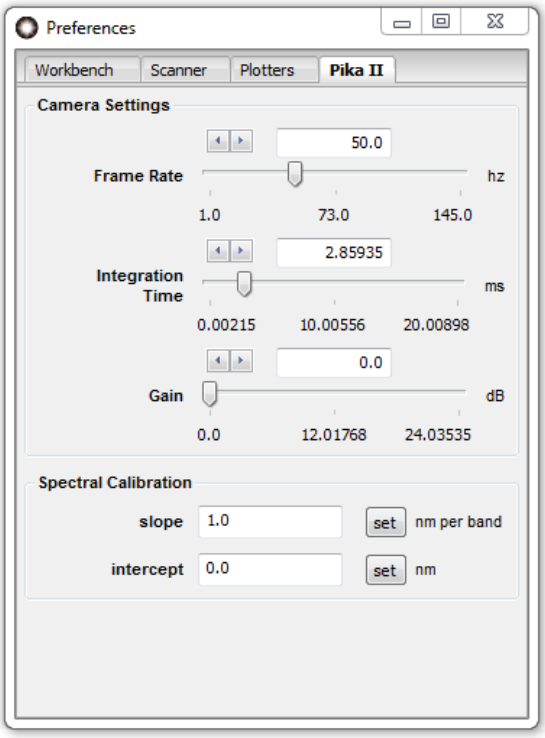


Fig. 5.2: **Pika II imager tab in Preferences window**

5.1.1 Basic Camera Controls

Integration Time (also known as **Exposure Time**) is the duration of data acquisition for each individual line image.

If you use the **Auto Expose** button in the toolbar , the integration time will be adjusted automatically for your lighting conditions such that there are no saturated pixels in the image. Make certain the reflectance calibration tile is being imaged when using this option.

Frame Rate is equal to the number of images acquired each second, and limits the maximum integration time (Max Integration Time (seconds) = 1.0 ÷ Frame Rate). If you do not have enough light, decreasing the frame rate will enable a longer integration time.

Gain is a factor which increases the signal, but at the expense of Signal-To-Noise ratio, in units of dB. We recommend that you operate at low Gain values (preferrably zero) to minimize noise. However, for situations where you need to operate at high frame rates and need more signal, you may increase the Gain manually. The **Auto Expose** button in the toolbar will always set the gain to zero. (Note: The Pika NIR camera does not have a Gain control tool.)

Hint: When adjusting the camera settings, you may observe the effects of your adjustments in live view. To do this, click on the focus button.

Note: If you observe broken or torn images when recording data at high speeds, you may be able to eliminate this problem by collecting data at slower speeds. These errors may occur due to a number of shortcomings within the data acquisition system including limitations in disk-write speed, chipset, motherboard bus speed, and CPU speed.

5.1.2 Slope and Intercept Calibration Values

As discussed in Chapter 3 (*System Assembly*), the **slope** and **intercept** values should be input when you first set up your hyperspectral imager. The values are provided on a paper calibration sheet with your imager. Contact Resonon support if you have lost your slope and intercept values. Once these values are set they should not be changed.

5.1.3 Auto Exposure Parameters

Spectronon features a user-configurable auto exposure algorithm that can be controlled in the camera settings tab. Two parameters are available: **Target Brightness** and **Target Pixel**. During the auto exposure routine, Spectronon calculates the histogram of pixel brightness of each frame. It then iteratively adjusts the integration time such that the histogram percentile represented by the **Target Pixel** parameter is at least as bright as the **Target Brightness**, but is not saturated. For example, if the **Target Pixel** is equal to 100% and the **Target Brightness** is equal to 3890 (the default settings for 12 bit cameras), the algorithm will attempt to find an exposure time such that the brightest pixel has a brightness of at least 3890 but less than 4095 (saturation). Similarly, if the **Target Pixel** = 95%, the pixel that represents the 95th percentile of total brightness will be targeted in each frame, instead of the brightest pixel. Selecting a **Target Pixel** less than 100% may cause the brightest pixels to be saturated, but may be useful for achieving desired exposure when features of interest are not the brightest in the field of view). For typical conditions, **Target Pixel** is set to 100% and **Target Brightness** is set to 95% of the camera's bit depth.

5.1.4 Windowing and Binning of Hyperspectral Data

Windowing refers to using a reduced subset of the available data, which reduces overall data volumes. Windowing can occur in both the spectral and spatial dimensions. Both the Pika XC and Pika NIR have windowing capability, while the Pika II does not. For the Pika NIR, windowing can also be used to eliminate portions of the focal plane array not utilized by the imager, which are the wavelengths outside the 900-1700 nm range.

Binning involves combining data from multiple adjacent pixels into a single data point. Binning reduces overall data volumes and increases SNR, at the expense of decreased resolution. Binning is possible in both the spatial and spectral dimensions.

There are two types of binning. One type is software binning, which occurs in Spectronon, and is available on all systems. The other type is hardware binning, which is only available on the Pika XC. In addition to decreasing data volumes and increasing SNR, hardware binning, when applied to the spectral dimension only, has the additional advantage of increasing the maximum frame rate. See the table below for details.

Pika XC Spectral Binning Parameters			
Spectral Bin Factor	**# of Spectral Channels**	**Spectral Resolution (nm)**	**Max Frame Rate (fps)**
5	240	2.4	153
6	200	2.9	181
7	170	3.4	213
8	150	3.8	241

Fig. 5.3: **Hardware Binning Parameters for the Pika XC**

On the Pika XC, it is preferrable to bin with hardware than with software, which will utilize fewer CPU resources.

Software binning is available on the **Scanner** tab in the **Preferences** window, see section 7.3.1 below. Windowing and hardware binning controls are accessed through the appropriate imager tab in the **Preferences** window. The **Start Spectral** and **End Spectral** values determine which spectral bands are collected. The **Start Spatial** and **End Spatial** values determine which spatial bands are collected.

GigE Connection Setup

The Pika II imager with GigE connection requires some additional steps for complete setup. If you purchased your computer from Resonon, these steps may have already been completed for you.

First, enable jumbo packets for your network adapter in Windows. In Windows, click the start menu, right click on *My Computer*, and clicke the *Manage* option. Click on *Device Manager* in the left column, click on *Network Adapters* in the middle column, right click your wired network adapter and choose *Properties*. In the *Advanced* tab click *Jumbo Packet* and set the Value to ~9014 Bytes. Click *OK* and close the *Computer Management* window.

In Spectronon, two additional settings will appear under the subgroup "GigE Properties" in the Pika II tab under the **File** → **Preferences...** menu: *Packet size* and *Packet Delay*. These settings allow for management of bandwidth between the camera and the controlling computer. The settings selected here will, in part, determine the achievable framerate. Larger packet sizes require less CPU usage to transfer a given amount of data from the camera to the computer than small packet sizes. The maximum packet size is limited by your network adapter, any other network hardware, and the camera. Typically, the largest available packet size is recommended.

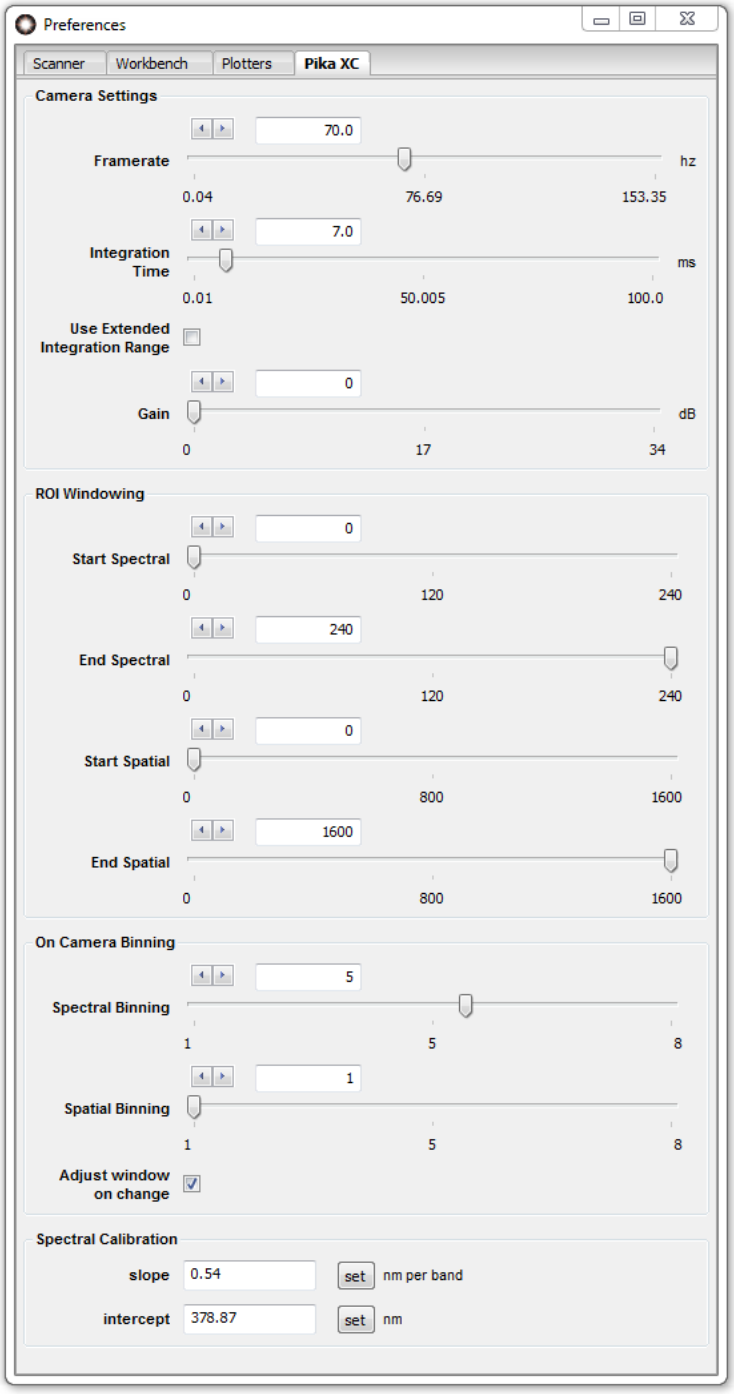


Fig. 5.4: **Pika XC imager tab in the Preferences window**

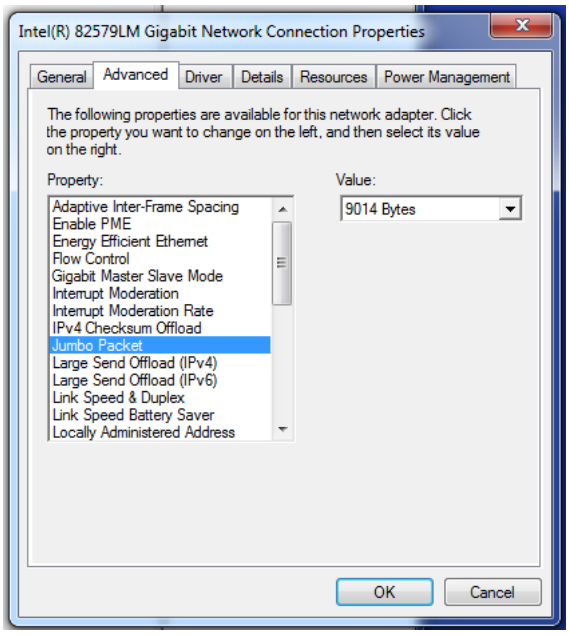


Fig. 5.5: **Device Manager - Network Adaptor - Jumbo Packets**

The packet delay sets the gap between data packets transmitted to the computer from the camera, and allows for time for the CPU to process the data it has received before new data is transmitted. If your scans result in dropped frames, try increasing the packet delay (which will impact the maximum framerate achievable).

Typically problems with these settings will manifest as image inconsistency errors, which are reported in the bottom status bar as a scan progresses. See the figure below for an example warning.

Inconsistency warning. Try increasing packet delay.

5.2 Stage Controls

You can move the stage manually by clicking on the **Jog Stage** buttons, located on the tool bar of SpectrononPro. The buttons will move the stage incrementally in either direction. Use the buttons to center the stage underneath the Pika imaging spectrometer.

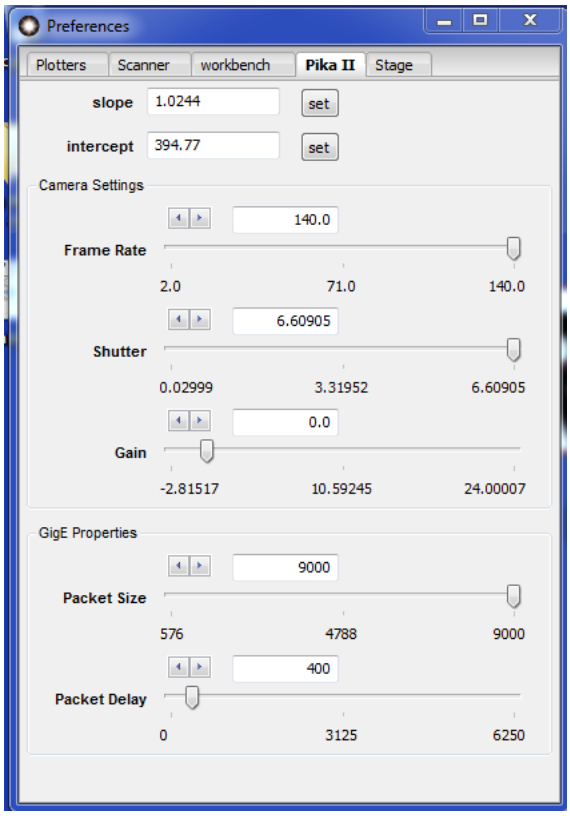


Fig. 5.6: **Pika II imager tab of the Preferences window**

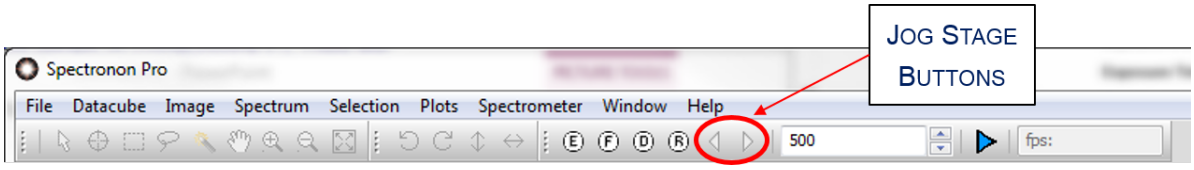


Fig. 5.7: **Jog Stage buttons located on the tool bar**

Further control of the stage can be controlled from two windows: the **Stage** tab in the **Tools Panel**, and the **Preferences** window (accessed by **File** → **Preferences**).

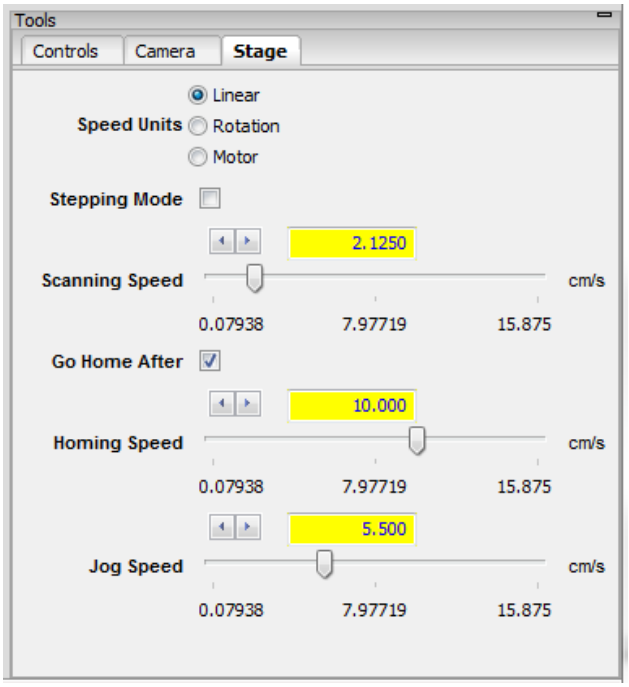


Fig. 5.8: **Stage tab in Tools Panel**

Speed Units is a setting used for different types of stages. **Linear** is used for the standard linear translation stage that is installed on most benchtop systems. **Rotation** is used for a tripod-mounted rotational scanning stage, typically used in outdoor applications. **Motor** displays speed in "motor pulses per second."

Stepping Mode controls the way the stage moves in relation to the imager, and is ideal for very slow scanning, very long integration times, or to guarantee there is no motion blur in your scan.

Scanning Speed is the linear speed of the stage during a scan.

If the **Go Home After** box is checked the stage will return to its starting position after a scan. The speed at which the stage returns to its original position is the **Homing Speed**, and the **Jog Speed** is used for the **Jog Stage** buttons, described above.

5.3 Additional Controls

5.3.1 Scanner Preference Tab

The **Scanner** tab allows you to set the binning options for recording your datacubes. These options are useful for situations where you wish to reduce the size of collected datacubes or where you wish to improve your signal-to-noise ratio.

The **Spectral Bin** option indicates how many spectral channels to bin. For many situations, this is your best choice to reduce the size of your datacubes and to improve your signal-to-noise ratio, as the spectral resolution is often higher than needed. With the Pika II, a spectral bin of 3 will result in 80 spectral bands (240 / 3).

The **Spatial Bin** option indicates how many spatial channels to bin. This option is typically useful when imaging an object known to be largely homogeneous.

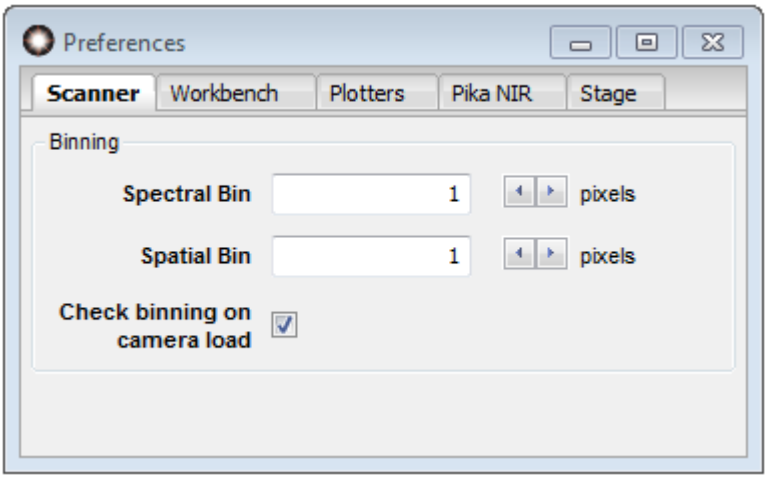


Fig. 5.9: **Scanner tab in the Preferences window**

Note: The Bin options listed above performs an average of the binned pixels. This allows you to directly compare binned and un-binned data.

5.3.2 Workbench Preference Tab

The **Selection** options relate to visualizing selected regions of datacubes.

The **Floodfill** options relate to the **flood fill** (wand) tool located on the menu bar, which is used to select a contiguous region of spectrally similar pixels. After selecting a region with either the **marquee** or **lasso** , right-click to reveal a pop-up menu with several options. Holding control while selecting ROIs allows you to append to the existing selection. **Metric** options are either Euclidean or Spectral Angle Mapper (SAM), and **Tolerance** is a unitless parameter that determines the sensitivity of **Floodfill**.

Default Image and **Open Cubes** pertain to opening existing datacubes. The options for **Default Image** are to open the image from your datacubes in a Red Green Blue (RGB) representation, or as a single grayscale image. The **Open**

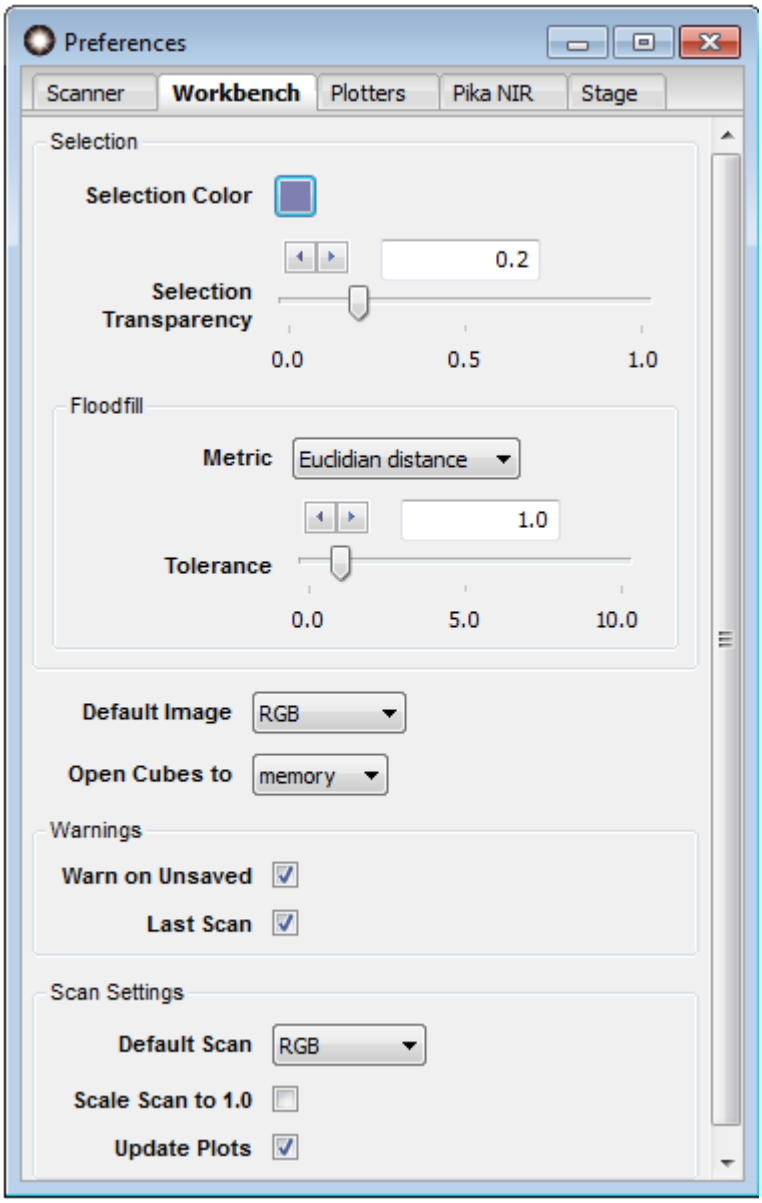


Fig. 5.10: **Workbench tab in the Preferences window**

Cubes options are either to memory or disk. The memory option is faster, but may not work well for computers with limited memory.

When checked, the **Warn on Unsaved** option will provide you with a warning message that you have not saved your last scan when closing the software. Similarly, when clicked, the **Last Scan** option will provide you with a warning message that your previous scan will be overwritten by your next scan if you have not already saved it.

The **Default Scan Image** option allows you to set whether your scanned image is displayed as a Red Green Blue (RGB) image or as a grayscale image. This setting does NOT impact the hyperspectral data itself; it only changes how the image of the data is presented in the image panel.

The **Recording** option allows you to record your scans into computer memory or directly to disk. Recording to memory is faster and preferred unless your data acquisition computer does not have sufficient memory for your scans. Be sure to click on the **Set** button if you change this setting.

The **Scale Scan to 1.0** option allows you to record reflectance as a floating-point number, with the brightness scaled to 1 for your reference sample. This saves your data as a standard reflectance value IF you use a reference sample with a reflectivity of 1. (Spectralon is a good material for this, and Teflon is a less expensive and reasonable substitute reference material, although less accurate.) If you use the **Scale Scan to 1.0** option, each channel of data will require 4 bytes instead of 2. Thus, the size of your datacube will double with Float as compared to not using this option.

When checked, the **Update Plots** option allows the plots to update during focusing and scanning. This option should normally be checked unless you have a sluggish data acquisition computer and you require more speed.

5.3.3 Spectrometer Menu

The **Spectrometer** Main Menu button provides alternative approaches to useful actions, as well as some timesaving options.

The **Reload Imager** option will restart the Pika imager without restarting SpectrononPro software. This option will save you time if you have, for example, accidently disconnected your Pika spectrometer or crashed the software.

The **Reload Stage** option will restart your scanning stage without restarting SpectrononPro software. This option saves you time if your stage needs to be restarted.

The **Auto Exposure** option allows you to let the software set the exposure setting for your lighting conditions. To use this, place an object within the field of view of your Pika, and then click on **Auto Exposure**. Alternatively, you can use the **Exposure** button from the toolbar.

The **Start Focus Tool** option will provide you with a live view from your Pika imager in the image panel. This tool is useful for adjusting the focus of your objective lens or checking the illumination of your system. You can also start the focus tool by clicking on the **Focus** button on the SpectrononPro tools bar, as described in Chapter 5.

The **Stop Focus Tool** option turns off the live view, which you need to do before you record a scan. The last live view will remain in a tab of the image panel. Another way to turn off the live view is to click the **Focus** button , which toggles live view off and on.

The **Record Dark Current Cube** is used to remove dark current noise from the imager. When you select this option you will be instructed to block the imager and then click **OK**. Another way to do this is select the **Dark Current button** on the SpectrononPro toolbar.

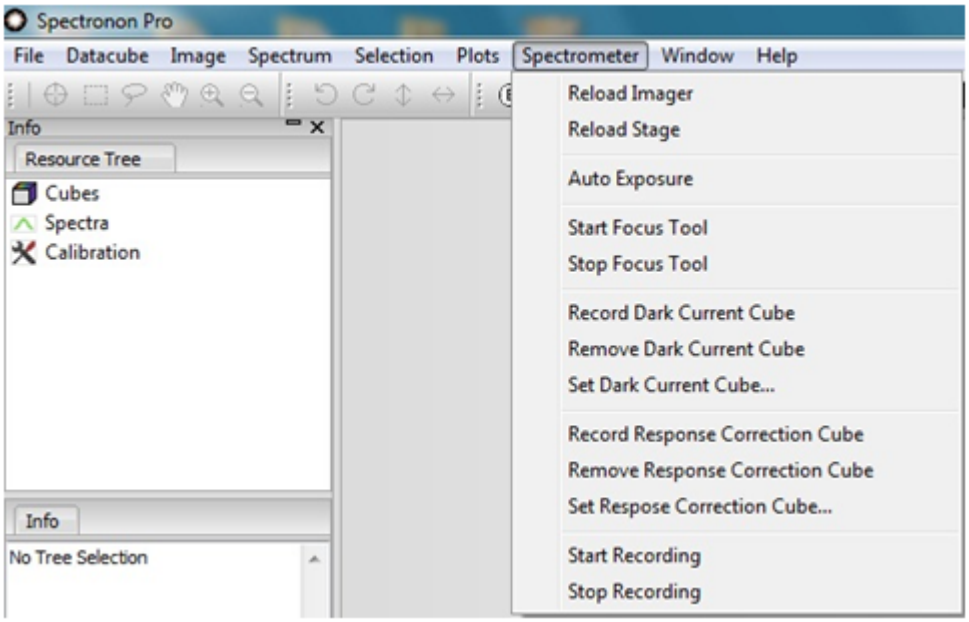


Fig. 5.11: **Spectrometer menu**

The **Remove Dark Current Cube** option will remove your dark current calibration cube. This will also remove the red check from the Dark Current button .

The **Set Dark Current Cube...** allows you to import a saved dark current correction cube. This option is rarely utilized.

The **Record Response Correction Cube** option allows you to use a reference material against which all measurements will be scaled. Typically this is done with a reference material whose reflectivity is approximately equal to 1. When you select this option, you will be instructed to place a reference material within the imager's field of view, and then click on **OK**. A small reference cube will then be recorded and used for scaling the data to the bit-level of the camera. (E.g., for the Pika II, 12-bits, or 4096, and for the Pika NIR, 14-bits or 16,384.) If you have selected **Normalize to Float**, as described above, the data with the scaled to 1 rather than the bit level of the camera. Another way to perform this function is to use the **Response Correction Cube** button on the SpectrononPro toolbar.

The **Remove Response Correction Cube** will remove the existing response correction cube. This will also remove the red check from the Response Correction Cube button .

The **Set Response Correction Cube...** allows you to import a saved reference cube. This option is rarely utilized, as one should generally record a correction cube regularly and with the current lighting to obtain accurate results.

The **Start Recording** and **Stop Recording** options will start and stop a scan with your Pika imaging spectrometer. This can also be done using the **Scan Button** on the SpectrononPro toolbar. The Scan Button will toggle Start and Stop functions.

CHAPTER

SIX

BASIC DATA ANALYSIS

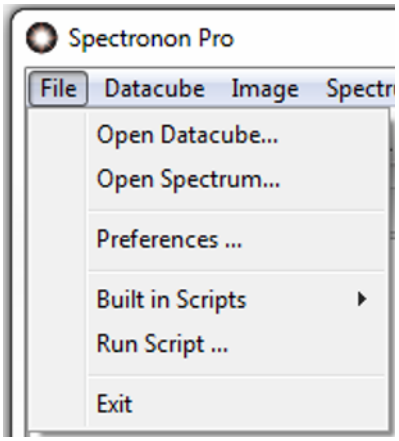
Spectronon provides visualization and manipulation capabilities for hyperspectral images. SpectrononPro software has all the features of Spectronon, but also enables data acquisition from Resonon's family of imagers. Spectronon software can be downloaded for free on Resonon's website http://www.resonon.com/. SpectrononPro comes bundled with any of Resonon's imaging spectrometers. For more advanced analysis capabilities, software packages such as ENVI® are available.

This chapter begins with basic operation of Spectronon, such as opening a hyperspectral datacube and viewing the data. A complete description of visualization tools is provided in **Chapter 8: Advanced Data Analysis**. **Chapter 9: Custom Data Analysis: Writing Plugins** discusses how to implement user-written algorithms into Spectronon, enabling custom data analyses.

References to "Spectronon" apply to "SpectrononPro" as well.

6.1 Spectronon Tools

To open a datacube, select **File** → **Open Datacube**.



This will open a dialog that allows you to browse to find your datacube. Select your datacube and click on **Open** to load your datacube. This will result in the following:

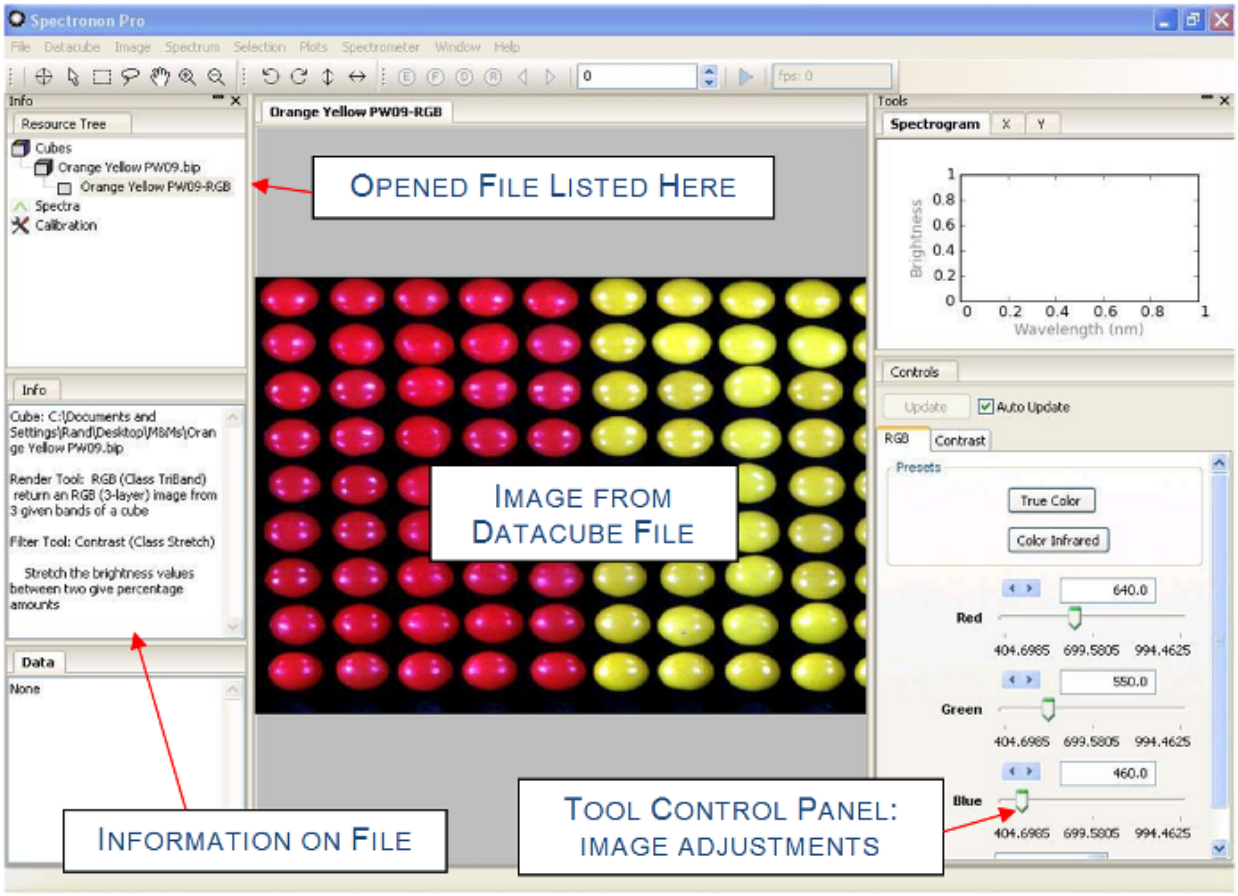
- An image of your datacube will appear in the image panel
- A listing of the open datacube will appear in the Resource Tree

- Tabs will appear in the Parameters window that allow you to change the image (more on this later)
- Header information on your datacube will appear in the information panel

---

Note: Spectronon can open any datacube with an ENVI© formatted header. This includes .bip, bil, and .bsq formats.

This chapter employs an example datacube of M&M® and Reese's® Pieces candies. (This datacube can be downloaded from Resonon's website at http://downloads.resonon.com/.) By default, the datacube is opened with a true color image of the data, which approximates the appearance of the object under normal lighting conditions by combining red, green, and blue wavelengths from the datacube.



With a few minutes of practice using the available tools, you will be able to manipulate and visualize hyperspectral data quickly and efficiently.

6.2 Zoom, Pan, Flip, and Rotate Tool

To zoom to a specific area of the image, select the *magnify* tool in the toolbar and the cursor will change.

Click the *magnify* tool in the image, and the view will zoom in. It is also possible to click and drag a selection within the image to zoom into the selected area.

To zoom out, select the *demagnify* tool and click anywhere in the image.

To zoom all the out and recover the original image, select the *original size* tool and click anywhere in the image.

The user may also zoom in and out using the mouse scroll wheel, if available.

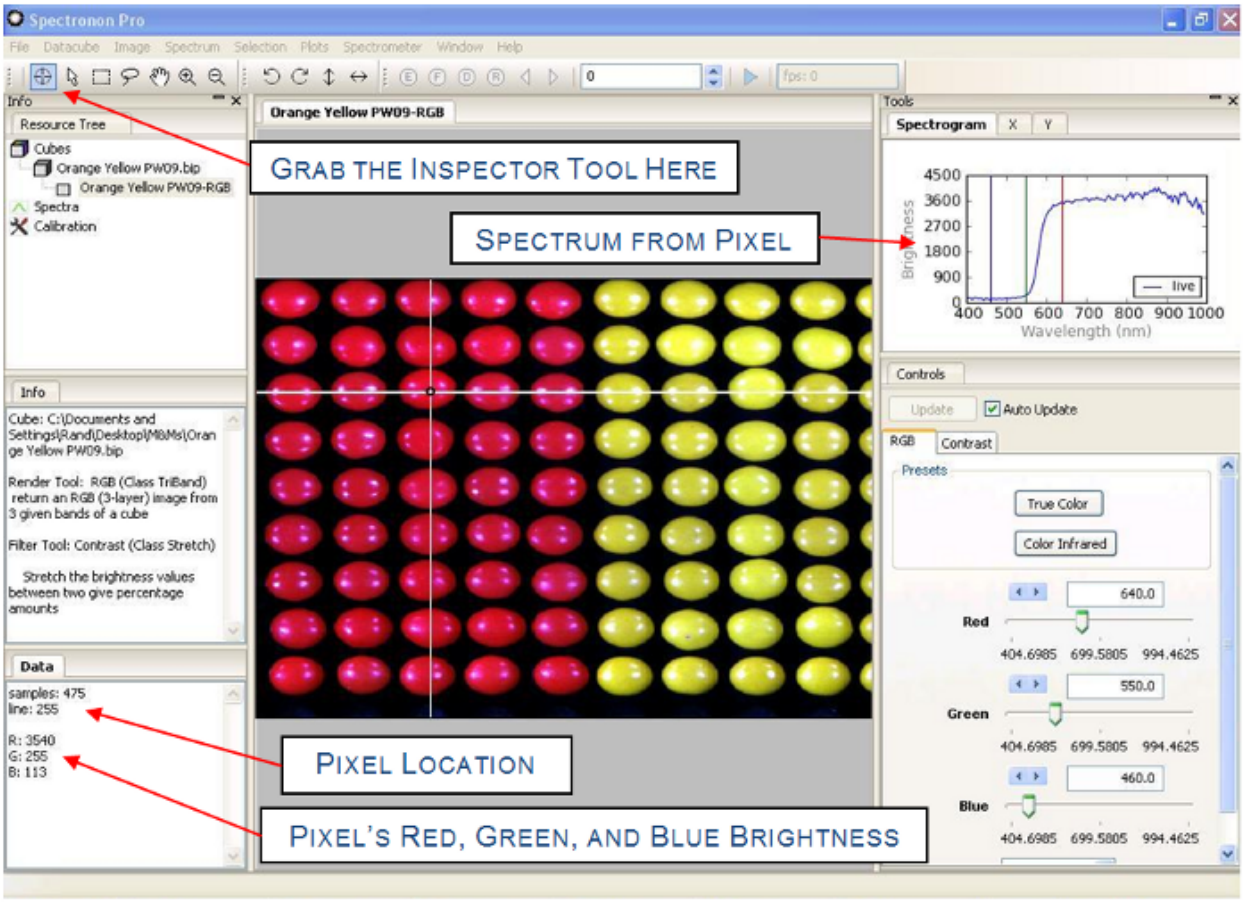
To pan the image while zoomed in, select the pan tool. Click and drag inside of the Image to pan.

Click these tool to rotate left, rotate right, flip vertically, or flip horizontally the image.

6.3 The Inspector Tool – Spectral Plots

The *inspector* tool allows you to see the spectrum associated with a pixel. Choose the *inspector* from the toolbar, and then click a point inside the image. This will:

- Plot the spectrum for the pixel in the spectrum plot panel
- List the pixel location (sample and line number) in the data panel
- List the red (R), green (G), and blue (B) brightness values in the data panel



Click on other pixels to see the spectra from other pixels, click and hold while dragging the *inspector* tool to update the *plot panel* continuously.

The red, green, and blue vertical lines in the spectral plot indicate the hyperspectral wavelength bands used to generate the current image.

6.4 Region Of Interest (ROI) Tools

It is often useful to consider a group of pixels within the image. The ROI tools enable this capability and provide a number of options. As will be seen later, the ROI tool is often used during one of the first steps in classifying different objects within a hyperspectral image.

To select a Region of Interest (ROI), select either the *marquee* , *lasso* , or *flood fill* (wand) tool from the menu bar. Click and drag a rectangle of interest with the *marquee* tool, or click and drag any closed shape with the *lasso*. The floodfill tool can be used to select a contiguous region of spectrally similar pixels. After selecting an area, right-click to reveal a pop-up menu with several options. Holding control while selecting ROIs allows you to append to the existing selection.

Hint: The selection menu is also available in the main menu.

The *floodfill* tool shows pixels that are spectrally similar to the chosen pixel. Spectral similarity is assessed with either Euclidean distance or Spectral Angle Mapper (SAM), along with a tolerance value. The user can set these options by

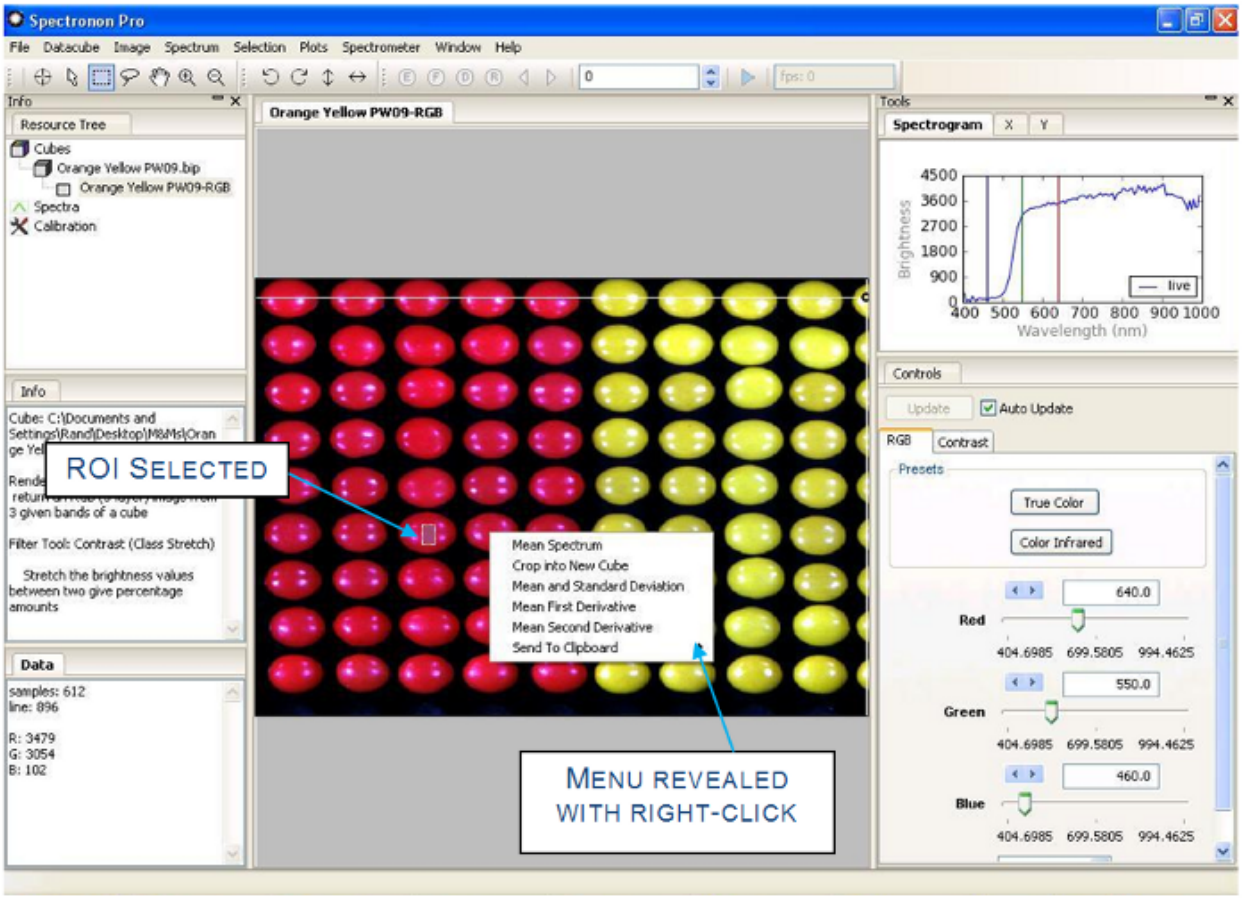
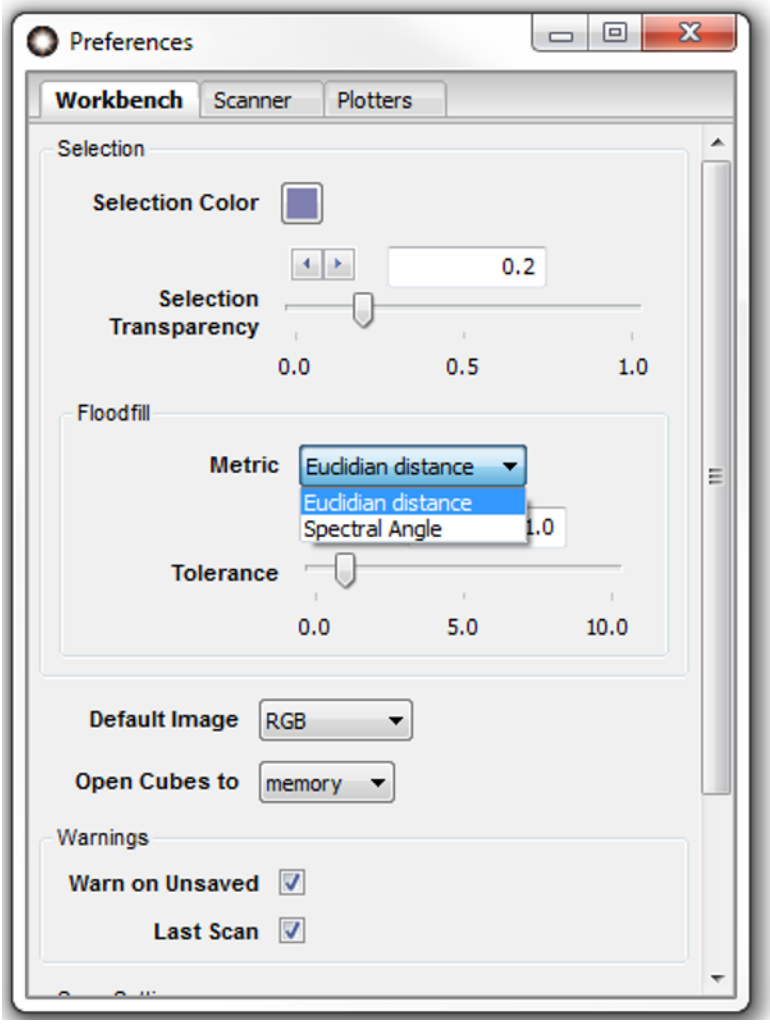
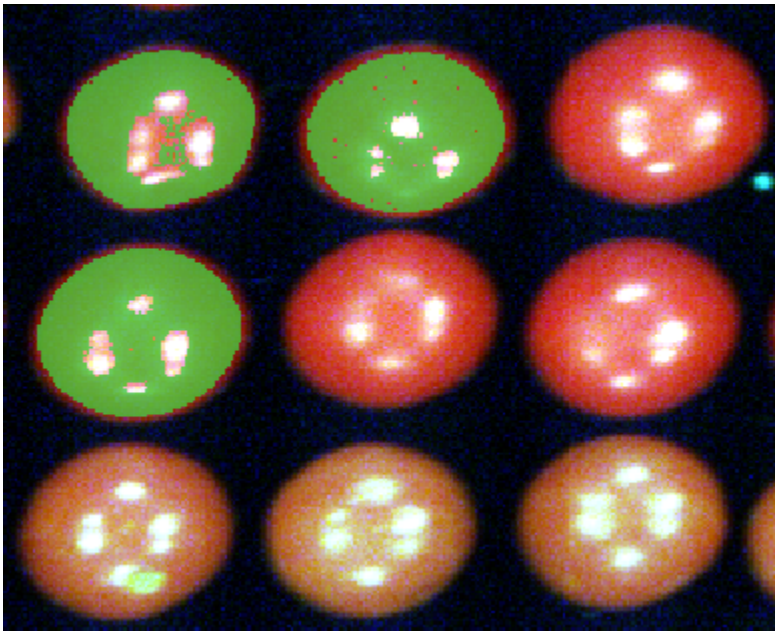


Fig. 6.1: **A small ROI on one of the red candies has been selected and a right-click has revealed the popup selection menu.**

accessing the *Workbench* tab from *File* → *Preferences* menu.

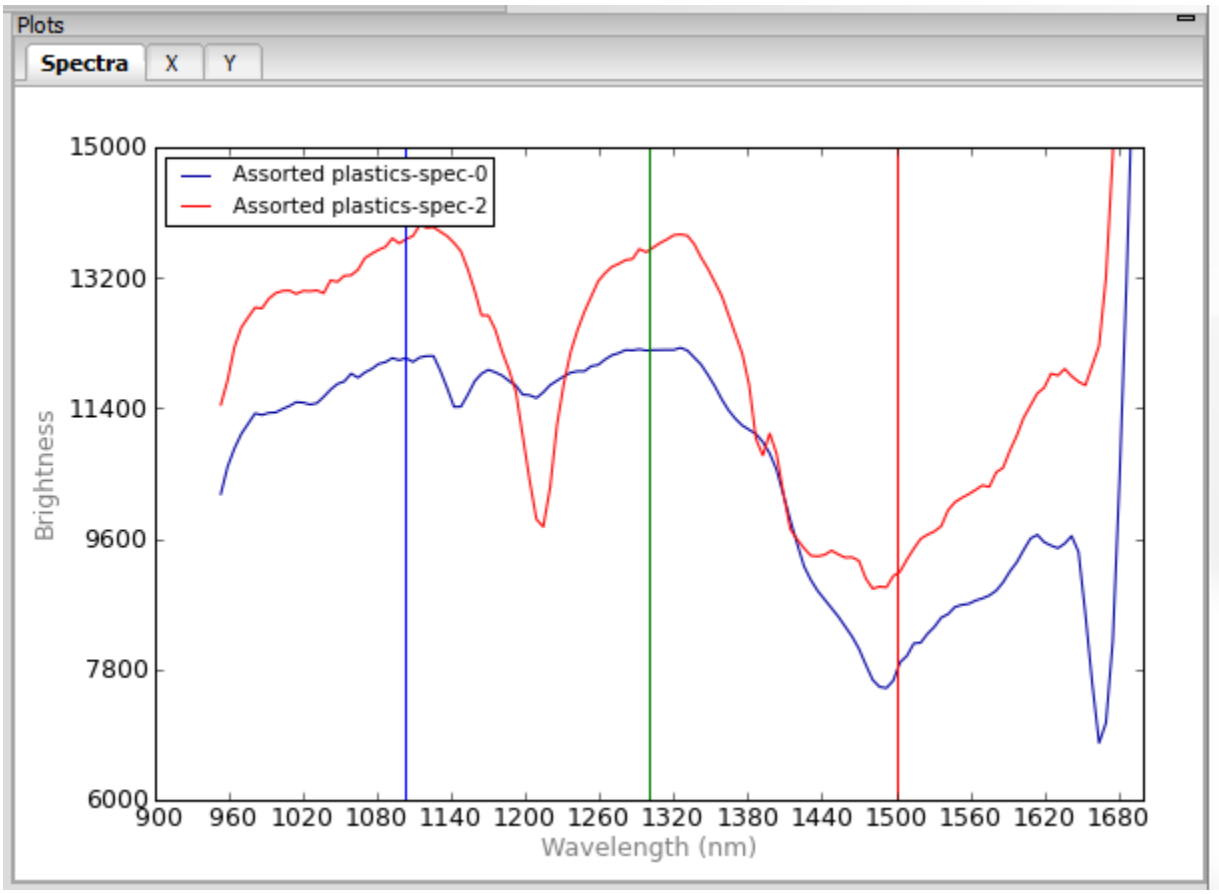


The use of the *floodfill* tool depends on an adjustable tolerance parameter. Floodfill operates on a representation of the datacube scaled from zero to one in each band. It calculates the Euclidean distance or SAM angle in spectral space between the clicked pixel and all contiguous pixels and expands the selection until the selected area contains all of the contiguous pixels for which the spectral distance to the clicked pixel is less than the selected tolerance. Increasing the tolerance will result in a larger selected region with greater spectral variability within that region (i.e. it allows pixels that are less similar to the clicked pixel to be included in the selection). Decreasing the tolerance will result in smaller selected regions with greater spectral similarity. As with the other selection tools, holding control while using the floodfill tool will allow a selection to be built up through multiple clicks of the tool.



An ROI consisting of multple parts of the image has been selected by using the floodfill tool several times.

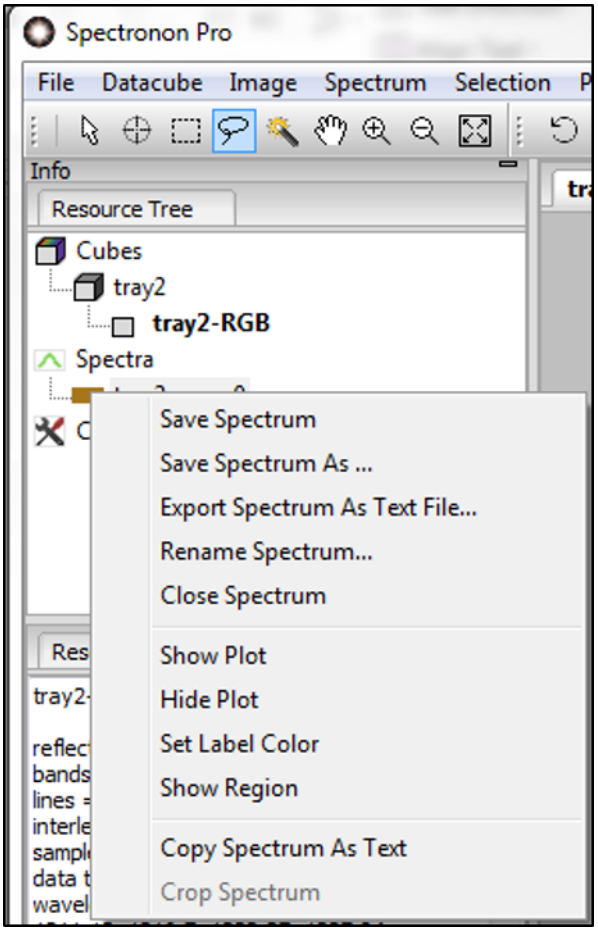
One of the most useful selection options is *mean spectrum*. (Descriptions for the other ROI options can be found in Chapter 4.) Selecting the mean spectrum option calculates the *mean spectrum* of all the pixels within the ROI area you selected and plots the result in the spectral plotter.



Individual spectra can be selected by clicking on the graph of the spectra. You can crop an individual spectrum by selecting it, selecting a rectangular region in the plot window, right clicking to bring up the menu, and selecting *crop spectrum*. You can also set the range of the plot by selecting *Plots* → *Spectral Plotter* → *Set Range*.

Hint: To examine the plots in more detail, you may resize the plot panel boundary by dragging the edges. Alternatively, click on the magnify tool, then click or drag in the spectral plotter to zoom in. The pan tool will allow you to pan within the spectral plotter as well.

Selecting *Mean Spectrum* creates a new entry in the *resource tree* under a new heading, *spectra*. Right-clicking on the spectrum in the resource tree will reveal a menu of options. Some of the most used options are listed below.



Save Spectrum

Change the name and save the spectrum as a file.

Set Label Color

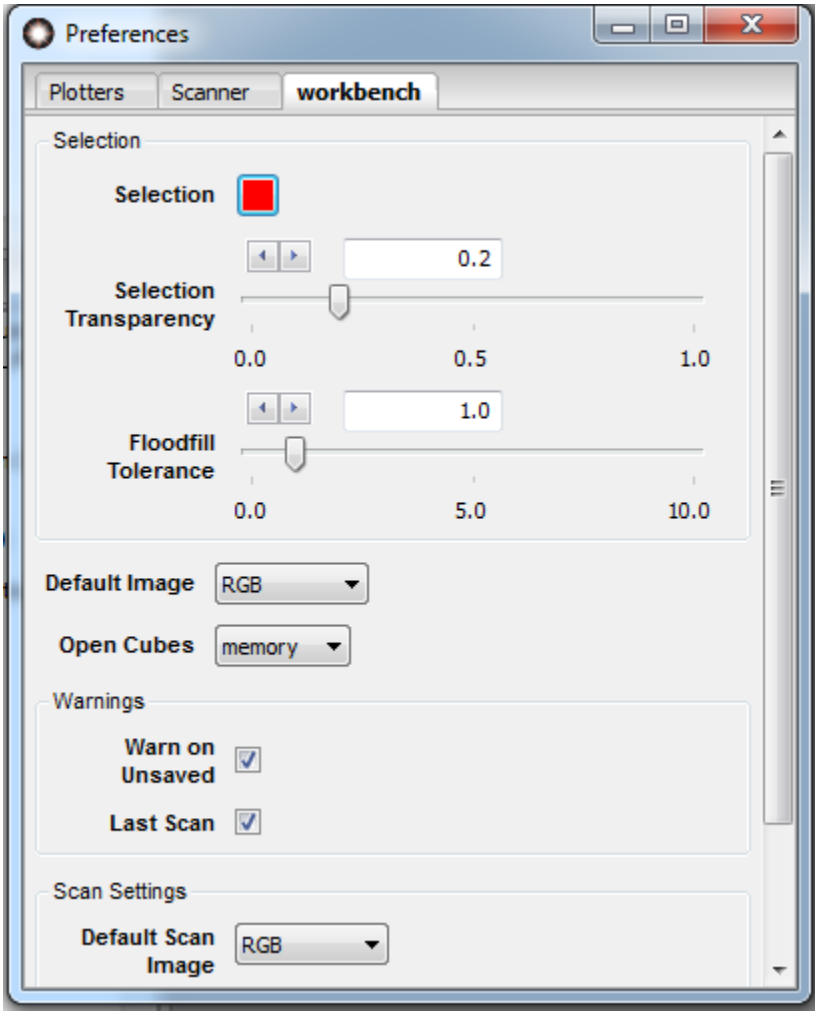
Open a color picker dialog to change the label color of the spectral plot, and as shown later, classification areas based on this spectrum.

Show Region

Show the originally selected area for the ROI in the current image. This option is useful after you have selected several ROIs.

The color and transparency of the selection can be modified by selecing *preferences* from the *file* menu. The selection options are located in the *workbench* tab of the preferences window. A selection transparency of 1 represents a

completely transparent selection (the selection will not be visible), while a selection transpatrency of 0 represents a completely opaque selection (the underlying render of the datacube will not be visible through the selection).

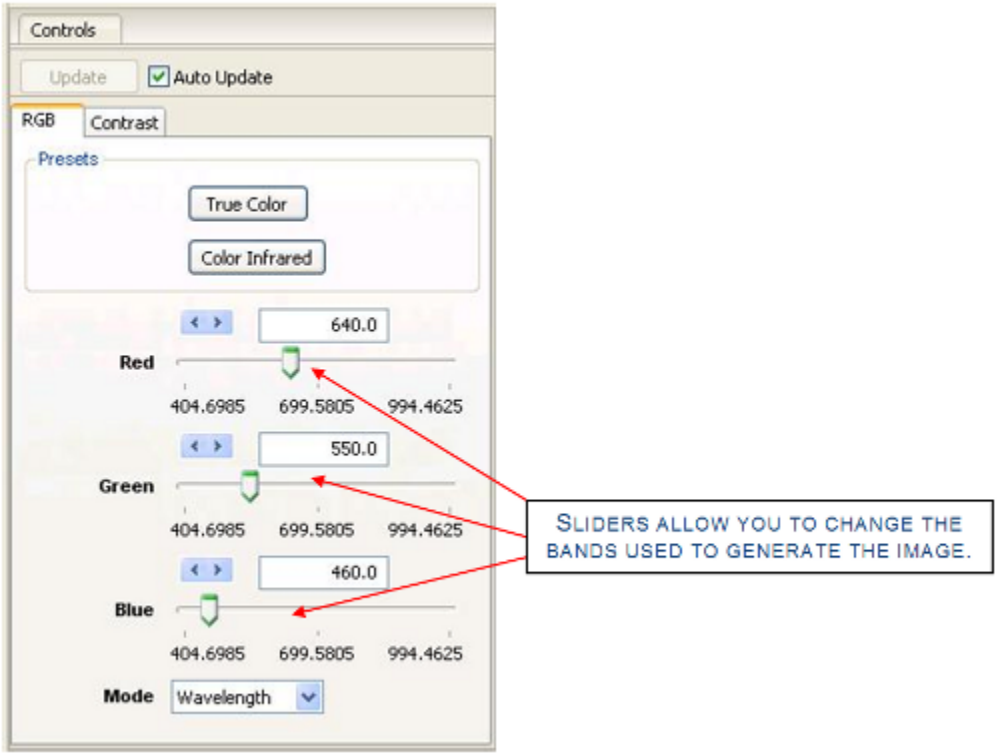


6.5 Image Visualization

Hyperspectral data can be visualized in far more ways than conventional color images. Image controls are provided in the tool control panel.

By default, the image is displayed in *True Color*, which means three representative bands are used to generate a red-green-blue (RGB) image, approximating how it appears to a human eye. The *Color Infrared* preset option provides a false-color RGB image with the red band set to an infrared wavelength. This option is useful for live vegetation datacubes.

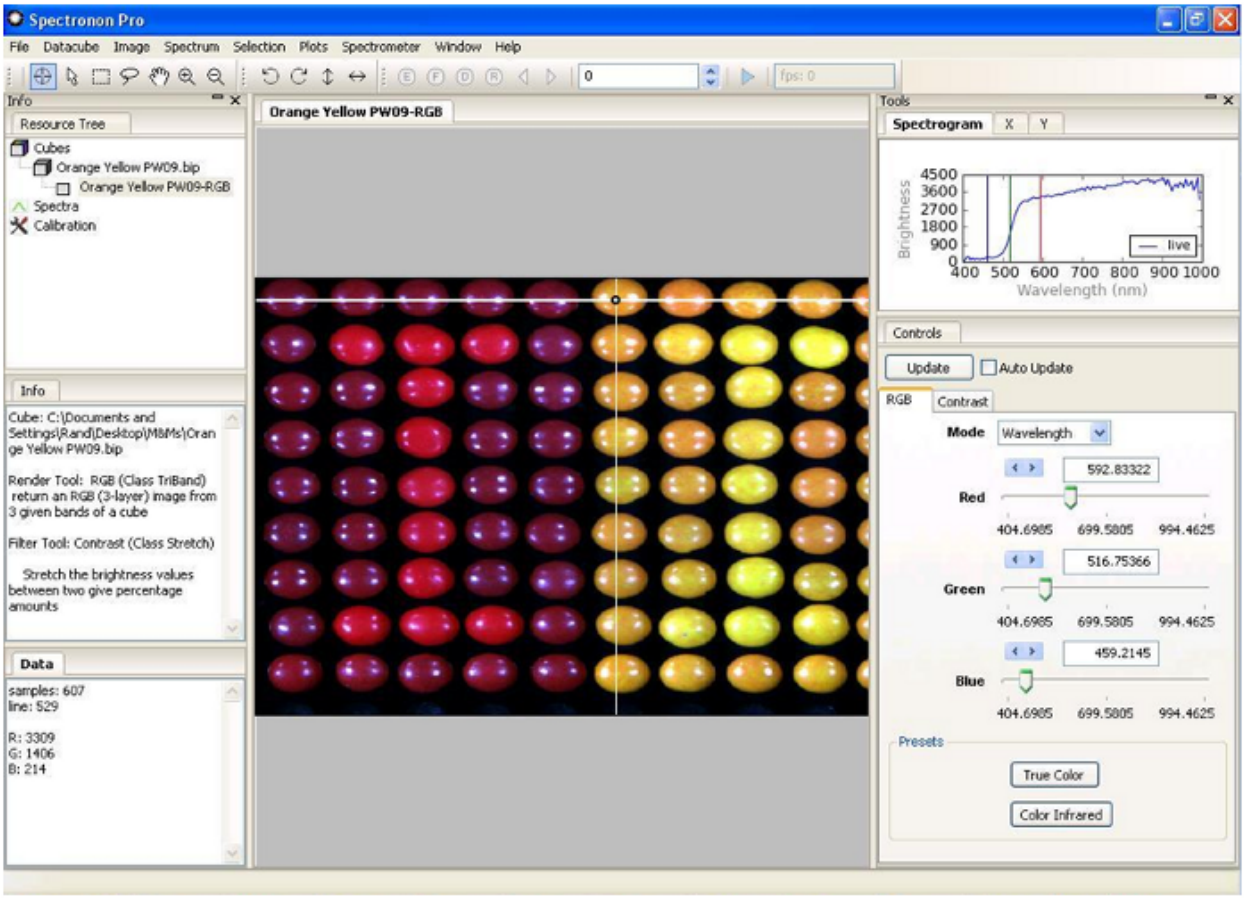
Any time you wish to restore the image to true color, simply click on the *True Color* button under *Presets*.



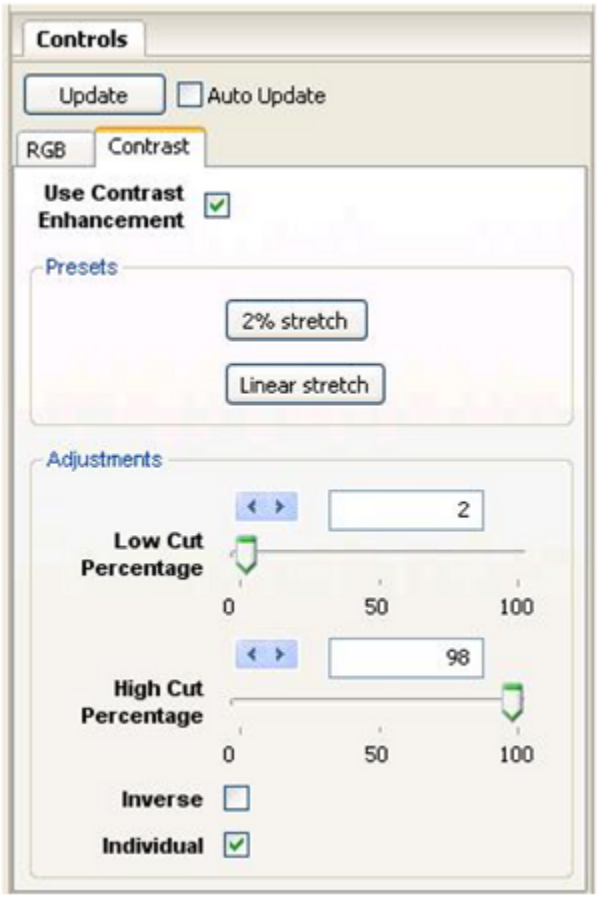
To generate false color images, use the sliders or arrows to change the wavelength bands used to create the RGB Image. This tool is often useful when trying to visualize specific spectral features associated with an object in your image. If *Auto Update* is not selected, click Update to generate the new image.

The *Mode* menu allows you to identify the band by wavelength (typically the most useful), or by band number.

As an example, of how false-color images can reveal interesting features, move the red slider to approximately 593 nm, and the green slider to approximately 516 nm, then click *Update*. This generates a new false-colored image, shown below, that reveals there are actually two kinds of red candy, and suggests there are two kinds of yellow candy – each candy type is positioned in the shape of an **"I"**. (In Chapter 3, a classification technique will show this more clearly.)



Note: The Red, Green, and Blue vertical lines in the Spectral Plot show the location of the bands chosen to create the false-color RGB image.



The *Contrast* tab in the tool control panel allows you to adjust the image contrast. If *Use Contrast Enhancement* is not checked, no image enhancement will be done and the tools in the Contrast tab will be not be active.

Note: Contrast enhancement does NOT change the hyperspectral data. It only changes the way the image appears.

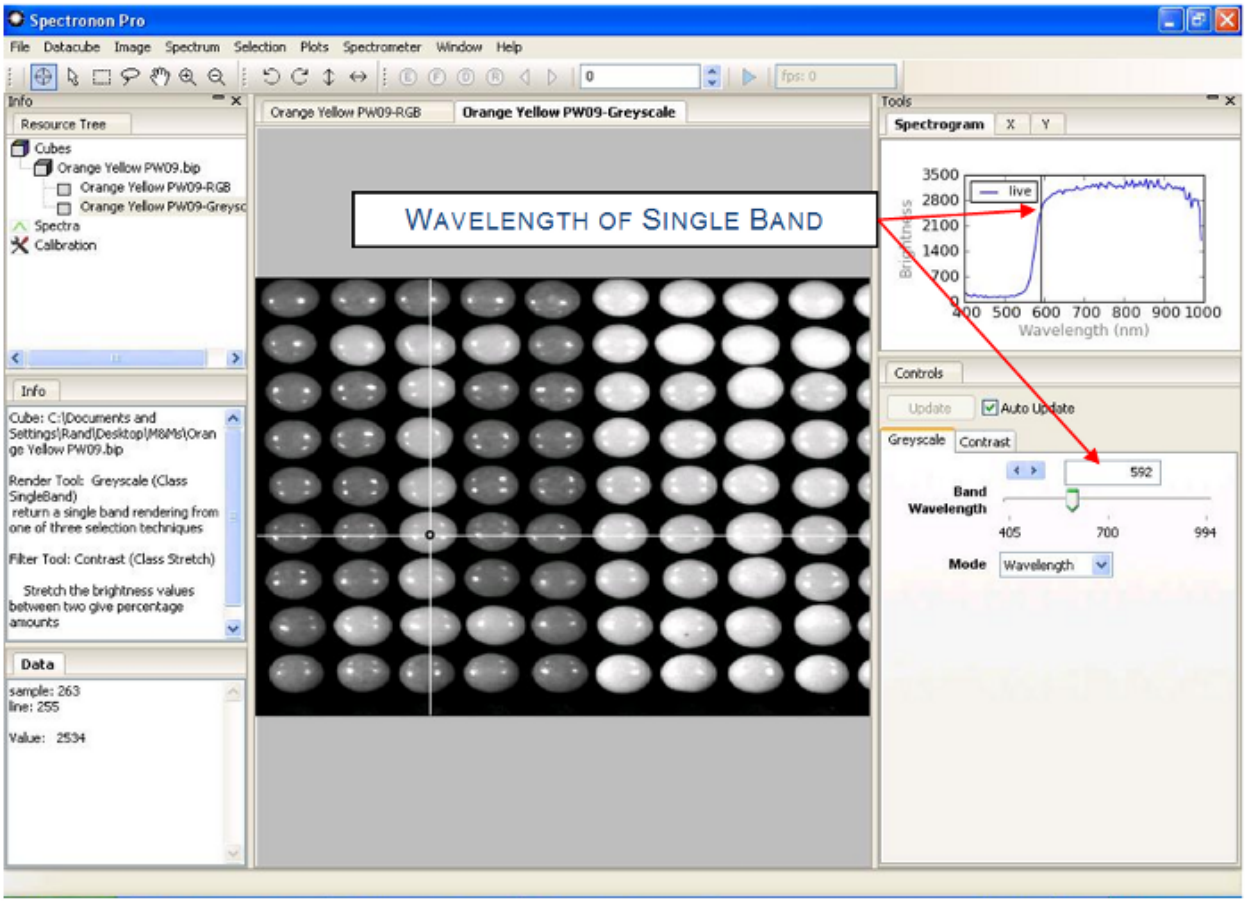
Generally, contrast enhancement is beneficial. The *2% stretch* is the default, and it sets the darkest 2% of the pixels in the image to a value of 0, and the brightest 2% of the pixels in the image to maximum brightness (255). This choice minimizes the impact of glare. You can customize the percentage of the dark pixels set to 0 and the percentage of the bright pixels set to 255 with the sliders. The *Linear stretch* option sets these percentages to zero.

The *Inverse* checkbox is useful if you wish to highlight dark pixels.

The *Individual* checkbox controls whether the brightness levels of the three image layers are considered all together or as individual layers.

It is often useful to view a single band in a standard grayscale (black-and-white) image to visualize the impact of a single spectral feature. To do this, go to the main menu and select *Datacube* → *New Image* → *Grayscale*.

The controls are similar to the RGB controls. If *Auto Update* is not checked, be sure to click *Update* after moving the slider to see the grayscale image for a new band.



As with the RGB images, a vertical line in the Spectral Plot shows the band you have chosen. Note that even though the image is from a single band, the Inspector and ROI Tools will continue to plot and operate on all wavelengths.

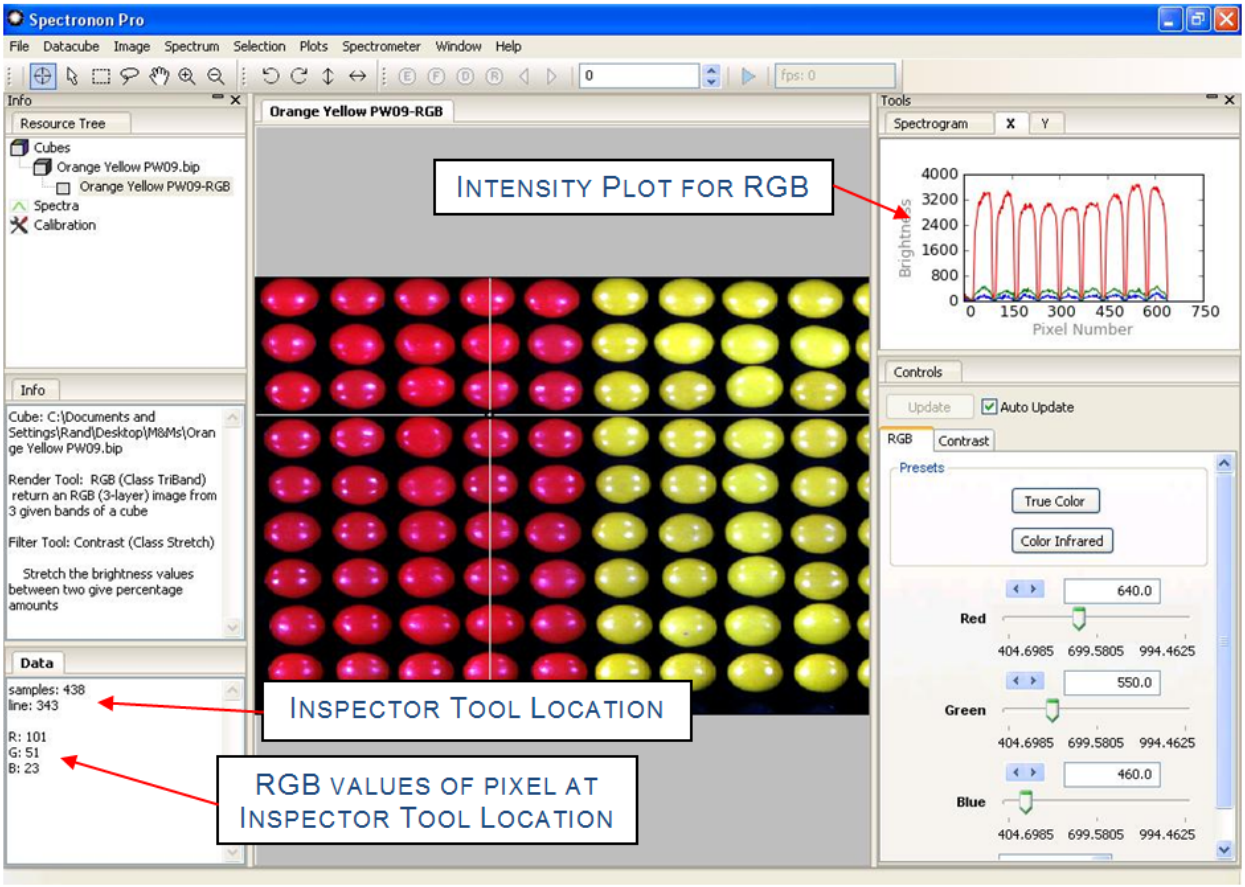
Hint: With Auto Update selected in the tool control panel, you can quickly scroll through single band images.

Warning: For large datacubes or slow computers, the *Auto Update* refresh rate may be slow.

6.6 Plot Panel

The plot panel allows you to visualize hyperspectral data graphically. This has already been seen with the use of the Inspector and ROI tools, but here we explore the plot panel in more detail.

The plot panel has three tabs: *Spectrogram*, *X*, and *Y*. These three tabs provide you with plots along the three axes of a datacube using the Inspector Tool , as shown below.



Clicking on the *X* and *Y* tabs in the plot panel accesses the corresponding cross-sectional plots. The plot will show the intensity versus position value for the RGB bands used to create the image or the Grayscale band if used with a grayscale image.

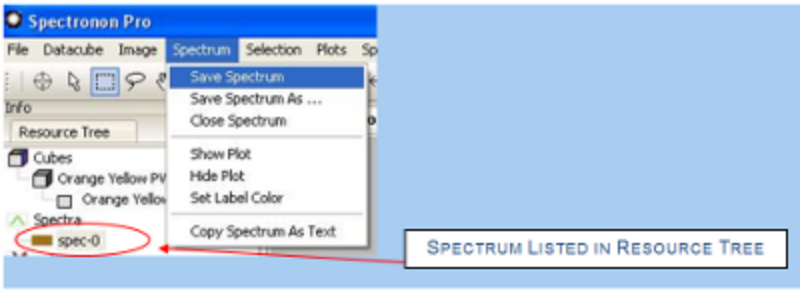
Note: The direction of *X* and *Y* depends on the orientation of your cube. Moving the *Inspector Tool* should reveal which axis you are plotting.

Hint: Use the *magnify tool* , the *demagnify tool* , and the *pan tool* in the spectral plotter to navigate and examine features.

6.7 Saving Spectra, Plots, and Images

Spectronon makes it easy to save the results of your work for further investigations or for making presentations.

To save a spectrum, click the spectrum you wish to save in the *Resource Tree*, and then you may either (1) use the *Spectrum* menu in the main menu, or (2) right-click on the spectrum in the *Resource Tree* to reveal the menu shown below. From this menu, select either *Save Spectrum* or *Save Spectrum As...* This will open a save dialog. Once saved, the new name will appear when the file is plotted in the spectral plotter, and the file can be re-opened for use in later sessions.



Select the menu option *Copy Spectrum As Text* to copy the data onto your clipboard, from which you can paste it into other applications such as Notepad and Excel.

To save a plot use the *Plots* menu as shown below. Select which plot you wish to save (Spectral, X Cross Section, or Y Cross Section), and then select *Save as Image* to save as an image or *Save as Text* to save the plotted data as tables in text file. Both options will pop up a save Dialog.

To Save an Image, select *Image* from the main menu, and then *Export Image...* This will pop up a save dialog.

6.8 More Information

Complete information about Spectronon Tools and hyperspectral classification can be found in the **Benchtop System Users Guide**, located at the product support page at the Resonon website: http://www.resonon.com/customer_support.html.

CHAPTER

SEVEN

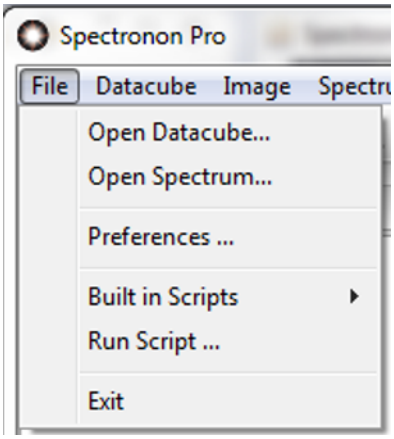
ADVANCED DATA ANALYSIS 1: GENERAL

A complete listing and explanation of Spectronon's data visualization tools is presented in this chapter. The organization of this chapter follows the main menu buttons.

7.1 File Menu

7.1.1 Open Datacube:

Clicking this option will open a window that allows you to browse and select the datacube you wish to work with. Only ENVI® compatible files will appear. Once selected, click Open, and the datacube will be loaded and an image will be generated in the image panel.



7.1.2 Open Spectrum

Clicking this option will open a window that allows you to browse and select a saved spectrum. To see how to save a spectrum, see Chapter 4: Basic Data Acquisition.

7.1.3 Preferences

Clicking this option opens a window that allows you to set a variety of preference options. Options associated with data visualization can be found under the Plotters and workbench tabs at the top of the window.

The Plotters tab opens a window that allows you to control the presentation of the plots presented in the spectral plotter. Additionally, this window also allows you to set the parameters for exporting plot data so it can be manipulated or plotted using other software tools.

The workbench tab opens a window that allows you to set the Default Image preference. One of the most useful settings is RGB, which generates a Red-Green-Blue image based on the values of three chosen hyperspectral bands, which you can choose in the tool control panel. Selecting the True Color button in the tool control panel produces an image that approximates the colors you would see looking at the object. Adjusting the sliders allows you to generate false-color images. Be sure to click Update after adjusting the sliders. Grayscale is another useful option, which presents a grayscale (black and white) image based on a single hyperspectral band. Again, you can adjust this band using sliders in the tool control panel. Be sure to click Update after moving a slider.

Hint: Check the Auto Update box in the tool control panel so you do not have to continually click Update.

You may find this option is too slow for large datacubes. You can also choose to open your datacubes in computer memory or your disk drive under the workbench. Generally, opening a datacube in memory is faster, but for large datacubes, or for computers with limited RAM, you may find that you need to choose the disk option.

7.1.4 Built in Scripts

Spectronon enables you to run scripts to best suit your application. A variety of scripts are built in to Spectronon, and can be found in **File** → **Built-in Scripts**.

7.1.5 Run Script

You may also run Python scripts you write. The Run Script opens a window that allows you to browse to your script. Additional information on how to write Python scripts for your particular need is provided in Chapter 10.

7.1.6 Reload Plugins

When writing or changing existing plugins, clicking this will make Spectronon aware of the changes.

7.1.7 Exit

Clicking on the Exit option closes Spectronon.

7.2 Datacube Menu

7.2.1 New Image

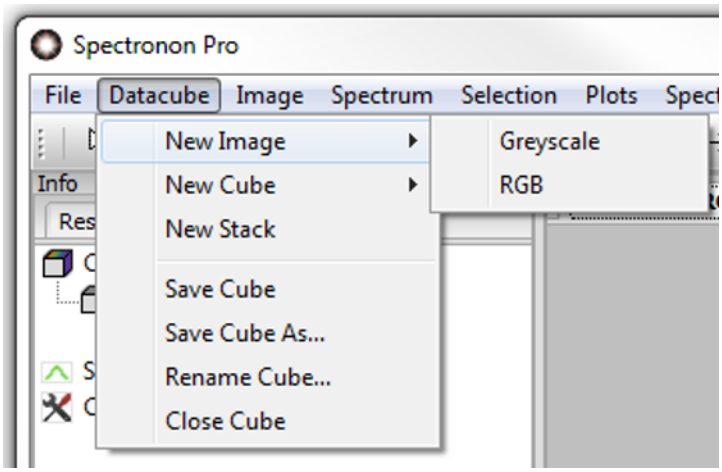
This option allows you to create a new image in the Image panel. When you click on New Image, a menu opens three options, RGB, Grayscale, and Raw Camera Data.

Datacube → New Image → RGB

Generates a color image based on three bands of your hyperspectral data. These bands can be chosen and adjusted using sliders in the tool control panel.

Datacube → New Image → Grayscale

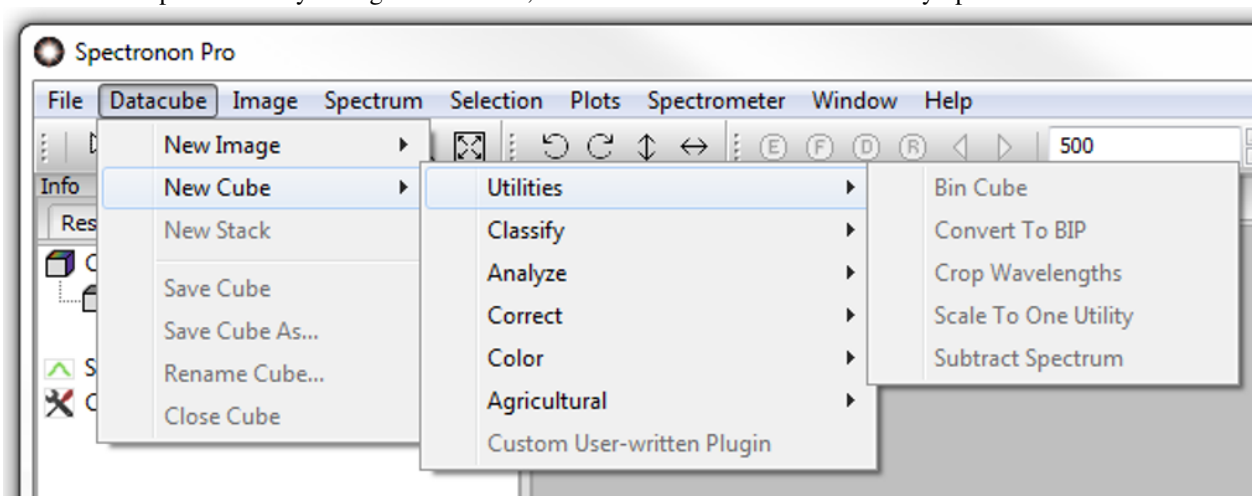
This produces a single-band image from a single hyperspectral band that can be chosen using a slider in the tool control panel.



7.2.2 New Cube

Utilities

The **Utilities** option allows you to generate a new, modified datacube from the currently open datacube.



Datacube → New Cube → Utilities → Bin Cube

Generates a new cube by binning spectral and/or spatial channels, which is often useful to either reduce the size of the datacube or improve the signal-to-noise ratio. This choice creates a new image in the Images Window and also generates a Bin Cube tab in the tool control panel with 3 sliders. The **Sample Bin** and **Line Bin** allow you to bin pixels along a spatial (x,y) axis. The Sample axis refers to the cross-track axis of the imaging spectrometer, and the Line axis refers to the along-track axis of the imaging spectrometer. The **Spectral Bin** slider allows you to bin spectral channels, and is likely the most useful of the binning options. After clicking **Update**, a new binned datacube is generated.

Note: The binned data are averaged and thus do not change significantly in amplitude. If you choose **Float Mode** the data will be rescaled with 1 set to the maximum bit level of the data. If your data have been scaled to a reflectance reference, you will find that glints can create Brightness values larger than one.

Datacube → New Cube → Utilities → Crop Wavelengths

Generates an image in the Image panel and a new tab with sliders in the tool control panel that allow you to crop wavelength bands by choosing a new minimum and maximum wavelength within the cube. Once chosen, click the **Update** button.

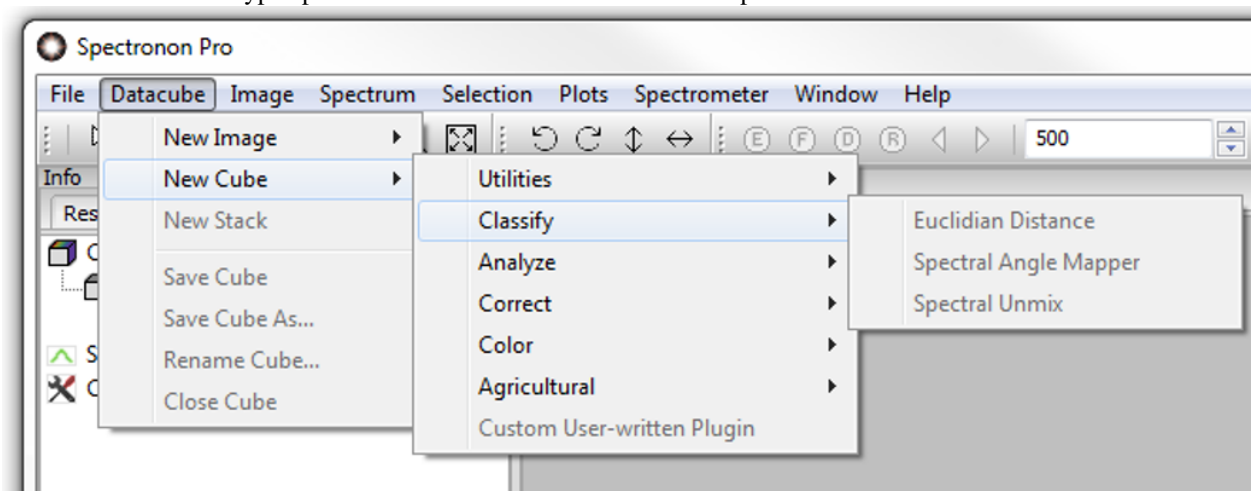
Hint: You may want to click on the RGB tab in the tool control panel to reset the bands used to create the image after cropping wavelengths.

Datacube → New Cube → Utilities → Subtract Spectrum

Generates a new cube by subtracting a background spectrum from all pixels in the datacube. This option is useful, for example, if you are monitoring fluorescent dyes and wish to subtract the background fluorescence of the substrate.

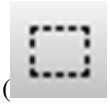
Classify

The **Classify** option allows you to generate classification maps of different objects within your hyperspectral data. More information on hyperspectral classification can be found in Chapter 8.



Some of the classification algorithms (SAM, Euclidean Distance, Spectral Unmix) use Spectrum objects as inputs, while the statistics-based classifiers (Logistical Regression, Discrimient Analysis, Support Vector) require datacubes

of the classification target. These are made selecting the ROI with the lasso or marqee tool, then selecting Create Cube from Selection.

When you select one of the classification options, (e.g. SAM), a new cube will be generated in the datatree panel. In many plugins, you will need to specify how many Layers you wish to classify (i.e., how many materials you wish to classify) – you will then need to select a Spectrum or Cube for each of these Layers in a pull-down menu. The Spectra and Cubes available in the pull-down menu can be generated either by using one of the ROI tools, marquee or lasso. ( or) or by loading a previously saved spectrum or cube.

Note: a detailed discussion of hyperspectral data classification is beyond the scope of this document. Spectronon provides some of the more commonly used algorithms. For more advanced algorithm capability, please see other packages such as ENVI®. Additionally, custom algorithms can be utilized with Spectronon using the scripting capability described in Chapter 9. The classification options provided are:

Datacube → New Cube → Classify → Euclidian Distance

This is a commonly used classification algorithm for hyperspectral data that is more sensitive to pixel brightness than SAM.

Datacube → New Cube → Classify → Spectral Angle Mapper (SAM)

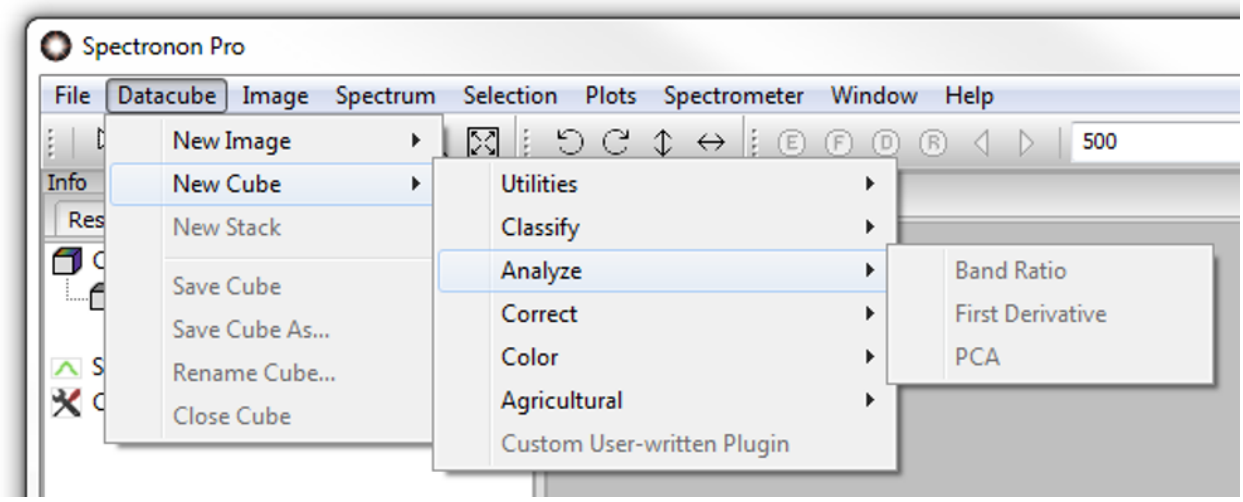
The SAM classification routine is described in detail in Chapter 6.

Datacube → New Cube → Classify → Spectral Unmix

Spectral unmixing deconvolves the signal in each pixel into a linear combination of known spectra.

Analyze

The **Analyze** option allows you to perform useful operations on hyperspectral data that will generate new datacubes based on applying analytical functions to your datacube.



Datacube → New Cube → Analyze → First Derivative

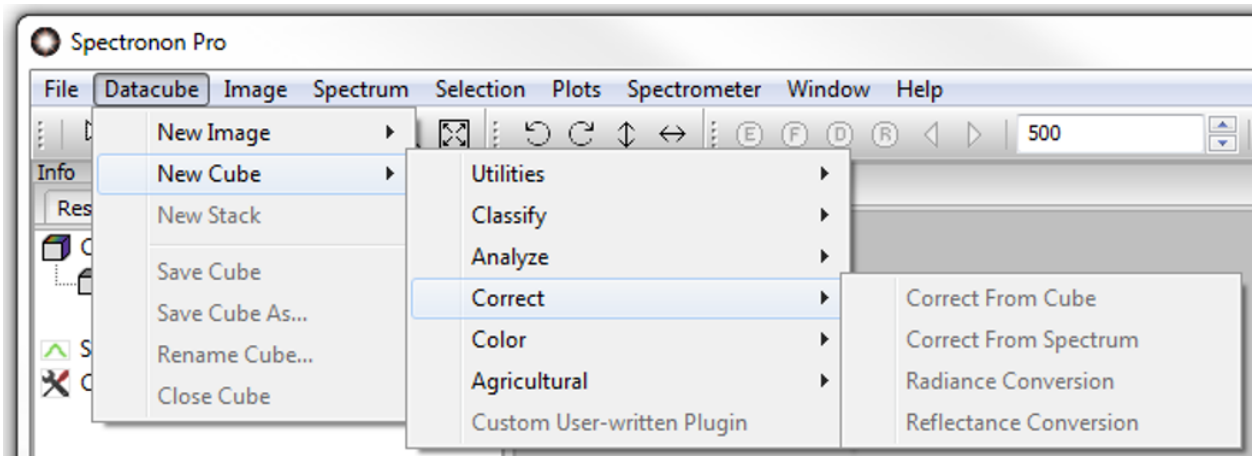
This tool generates a new datacube with the first derivative of the spectral curve for each pixel. A new image of the first derivative datacube is presented in the Image panel. To see the spectral derivatives, utilize the inspector tool , or use one of the ROI tools marquee or lasso (or). The first derivative curves appear in the spectral plotter.

Datacube → New Cube → Analyze → Principal Component Analysis (PCA)

A detailed discussion of PCA is beyond the scope of this document (see, for example, Wikipedia for a discussion). This tool generates a new datacube and image with the principal component values for each pixel. Additionally, a PCA tab appears in the tool control panel with a slider that allows you to select the number of Bands, or PCA components. To see the PCA component magnitudes, utilize the inspector tool , or use one of the ROI tools marquee or lasso. (or). The PCA magnitude curves appear in the spectral plotter. As with standard hyperspectral data, classification algorithms can be performed on the new PCA datacube.

Correct

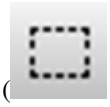
The **Correct** option allows you to correct a datacube to a measured reference. This tool allows you to convert your datacube to either radiance or reflectance values, as described below.



Datacube → New Cube → Correct → Correct From Cube

This tool allows the user to scale their datacube to another known reference datacube. When you select this option a **Correct From Cube** tab will appear in the tool control panel. In the **Correction** window select the cube you wish to correct to. Typically this is a datacube you have recorded of a uniform reference material. Once you click **Update** a new datacube will be generated that is scaled to the reference datacube. If you click **Float Mode**, data brightness will be scaled to "1" (and thus, if your reference has a reflectance of 1, your data will be in units of reflectance). Otherwise, the data will be scaled to the maximum bit level of your reference cube. (E.g., a pixel whose brightness matches a 12 bit reference datacube will be set to 212 = 4096.)

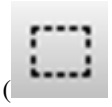
Datacube → New Cube → Correct → Correct From Measured Reference

The **Correct From Measured Reference** option allows you to scale your datacube to a known reference source. This option is useful for obtaining reflectance measurements during field deployments. Since this tool does NOT account for spatial variability (such as vignetting and slit variations), the data should be converted to radiance before using this plugin. The radiance conversion plugin is described below. To utilize the **Correct From Measured Reference** tool, it is best if there is a reflectance reference target located within your hyperspectral image. First, use one of the ROI tools, marquee or lasso, ( or) and generate a mean spectrum. Then select **Datacube → New Cube → Correct → Correct From Measured Reference** Use the Measured Reflectivity button to select the tab delimited file of measured reflectivty.

This will generate a new datacube scaled to brightness values of your Correction Spectrum. If you click **Float Mode** the brightness values will be scaled to "1." Otherwise, the data will be scaled to the maximum bit level of your datacube.

Datacube → New Cube → Correct → Correct From Spectrally Flat Reference

The **Correct From Spectrally Flat Reference** option allows you to scale your datacube to a spectrally flat reference source. This option is useful for obtaining reflectance measurements during field deployments. Since this tool does

NOT account for spatial variability (such as vignetting and slit variations), the data should be converted to radiance before using this plugin. The radiance conversion plugin is described below. To utilize the **Correct From Spectrally Flat Reference** tool, it is best if there is a reflectance reference target located within your hyperspectral image. First, use one of the ROI tools, marquee or lasso, ( or) and generate a mean spectrum. Then select **Datacube** → **New Cube** → **Correct** → **Correct From Spectrally Flat Reference** Enter in the percentage reflectivty (0-1) of the reference material.

If you click **Return Ints 0-1000** the values will be scaled to "1000" Otherwise, the data will be scaled to 0-1 and returned in floating points.

Datacube → **New Cube** → **Correct** → **Radiance Conversion**

This plugin requires a radiometric calibration datacube and will return radiance units of microflicks. To use, apply the plugin to the cube to convert and select the appropriate radiometic calibration datacube. If camera dark current is significant, an optional dark cube can be collected at the appropriate integration time and gain and utilized. Select this cube if necessary. Unlike some other plugins, the **Return Floating Point** checkbox will return the same values (microflicks) as the integer mode, but may be necessary for very bright scenes if radiance values above 65k microflicks are expected.

Datacube → **New Cube** → **Correct** → **Reflectivity Conversion**

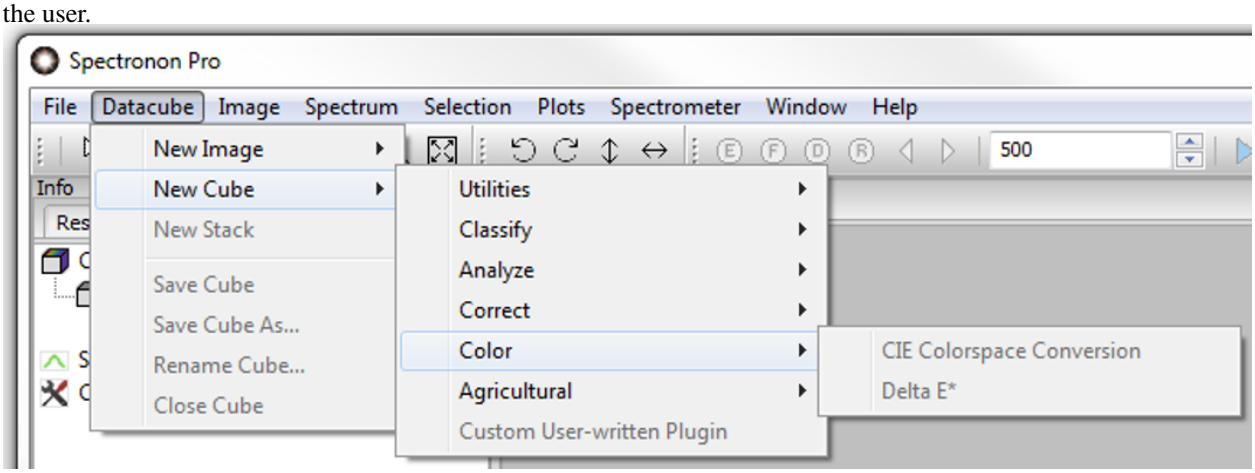
This plugin is intended for data collected with the Resonon airborne hyperspectral imaging system. It requires a spectral imager radiometric calibration cube, downwelling irradiance data relevent to the datacube being collected, and downwelling irradiance sensor radiometric calibration. Optionally, dark current data for the spectral imager and downwelling irradiance sensor can be used if the dark current for these systems is significant. The returned data will be in units of reflectivity from 0-1 for float mode or 0-1000 in integer mode.

Anomaly

Anomaly detection tries to determine anomalous outliers in a datacube that do not conform to the expected spectra. This is usually based on a datacube of 'clutter', which contains the expected spectra.

Color

The **Color** option allows you to transform your hyperspectral data into CIE colorspace, providing XYZ, xyY, and LAB values for each pixel. Additionally, you can determine the ΔE values for each pixel as compared to a standard set by the user.



To transform the hyperspectral data into **CIE Colorspace**, choose the pathway **Datacube** → **New Cube**→ **Color** → **CIE Colorspace Conversion**.

This will generate a new datacube with CIE colorspace values for each pixel as well as a new CIE Colors pace Conversion tab in the tool control panel. In this window you are provided with standard options for the Observer and Illumination. To see the Color space Values, position the inspector tool on a pixel, and its values will appear in the Data panel.

Note: A warning will appear if your datacube does not have all the spectral values needed to perform a CIE color space conversion (380 nm to 720 nm). Click "ok" and Spectronon will utilize the bands available to make the appropriate conversion based on the data available.

You can determine how close the pixels are to desired Color space values by using the ΔE option. To do this, you first have to perform the CIE Color space conversion, as described above, then follow the path **Datacube** → **New Cube** → **Color** → **Delta E***

This will generate a new image in the Image panel and a new tab in the tool control panel called **Delta E***. A slider appears in the tool control panel that allows you to select how many ΔE values you wish to compute. Below this, sliders appear that allow you to set your target LAB values. After setting your target LAB values, click **Update**. This will update your image. Use the inspector tool to see the ΔE values in the Data panel. To obtain a visual representation of the ΔE values, click on the **Threshold to Colormap** tab in the parameters window and adjust the thresholds.

Agriculture

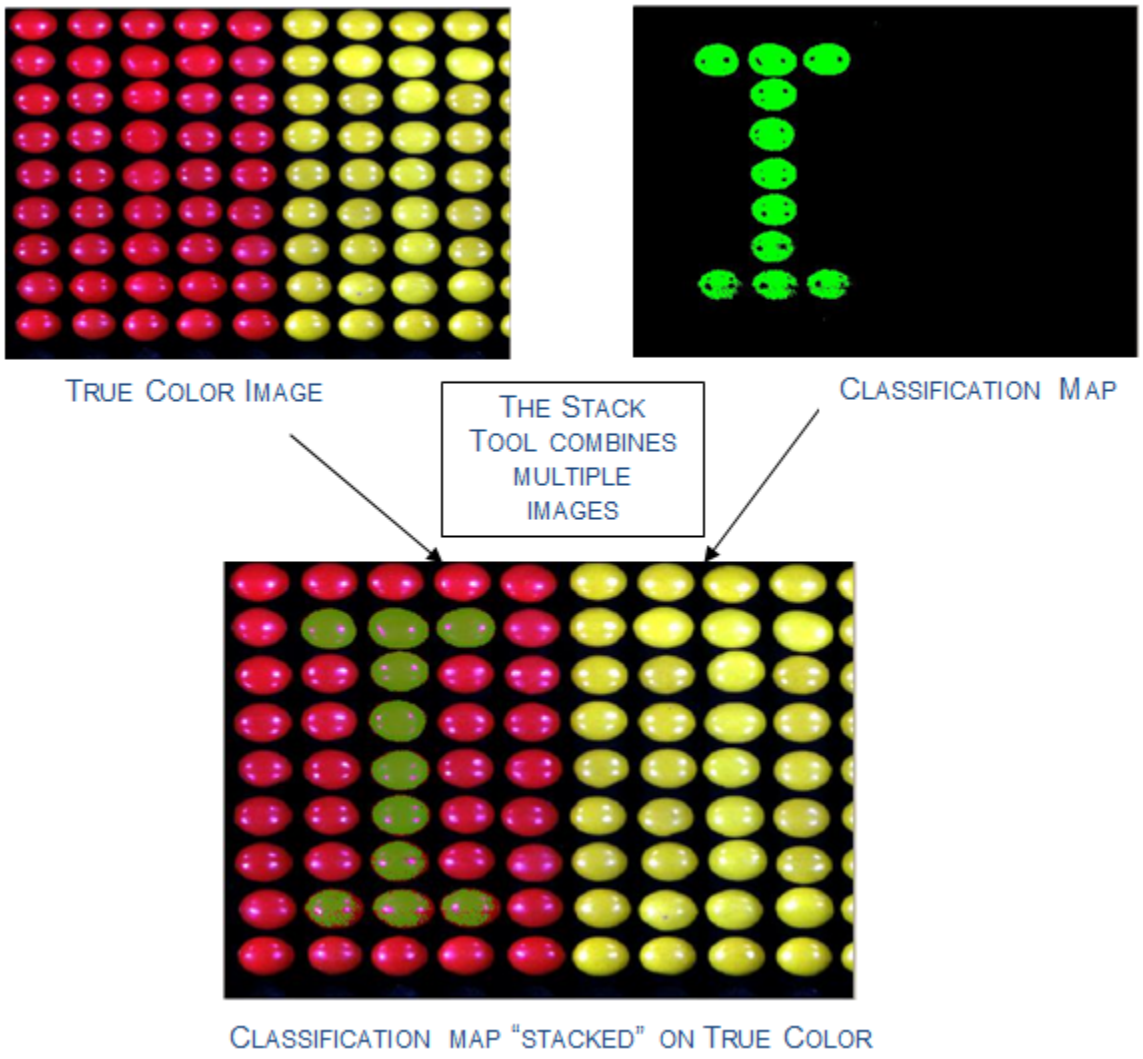
This option allows the user to create maps of Hyperspectral Vegetation Indices (HVI's). HVI's in Spectronon are discussed in greater depth in Chapter 9.

Clustering

Clustering tools are unsupervised classification routines that do not require any a priori information of a spectral image. They group the scene into clusters based on spectral similarity. Supervised classification routines can almost always produce better results, if scene knowledge available.

7.2.3 New Stack

This tool enables you to overlay multiple images. This tool is particularly useful for presenting classification results. For example, we may have a True Color RGB image of an object, such as shown below, as well as a classification map of one of the candy types. Using the Stack tool, we can combine these images to show the classification map on top of the RGB True Color image.

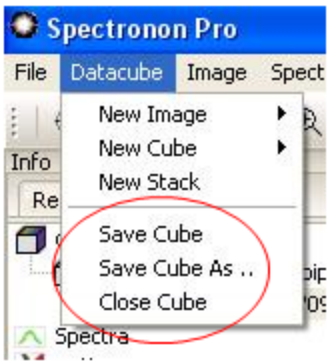

TRUE COLOR IMAGE
CLASSIFICATION MAP
THE STACK TOOL COMBINES MULTIPLE IMAGES
CLASSIFICATION MAP "STACKED" ON TRUE COLOR

To use the Stack tool, follow the path **Datacube** → **New Stack**. This will open a Stack tab in the tool control panel. Use the slider to select how many images you wish to combine (Stack height). Then click **Update**. Pull-down menus appear that allow you to select the images you wish to combine. A slider (Alpha) is available for each image that allows you to set its transparency in the combined image. Once set, click **Update** and your combined image will appear.

Tools for saving and closing datacubes are under the **Datacube** main menu option.

7.2.4 Save Cube

This tool will save the datacube currently open. Since many datacubes can be open at the same time, be sure to select the cube you wish to save in the **Resource Tree** before clicking on **Save Cube**. If the datacube has not been previously saved, a window will open that allows you to rename the datacube and save it in a file of your choosing.



7.2.5 Save Cube As

This tool allows you to save the currently open datacube under a new name or to a new location. Clicking on this option opens a window that allows you to name the datacube and save it in a file of your choosing.

7.2.6 Rename Cube

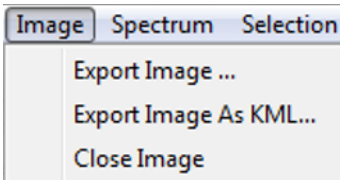
Use this to rename datacubes.

7.2.7 Close Cube

The tool closes the datacube selected in the Resource Tree. If this datacube has not been saved, a message will appear asking to confirm whether or not you want to close it. Clicking OK will close the datacube. Clicking Cancel will cancel the command so you can continue working with the datacube or save it.

7.3 Image Menu

The **Image** menu provides tools for saving or closing images generated from your datacube.



7.3.1 Export Image

Clicking on **Export Image...** will open a window that allows you to name and save the image file at a location of your choosing. Additionally, you can choose your preferred image format. Options include: TIFF, PNG, BMP, JPG, and GIF.

7.3.2 Export Image as KML

This allows the user to export an image in Keyhole Markup Language (KML) format.

7.3.3 Close Image

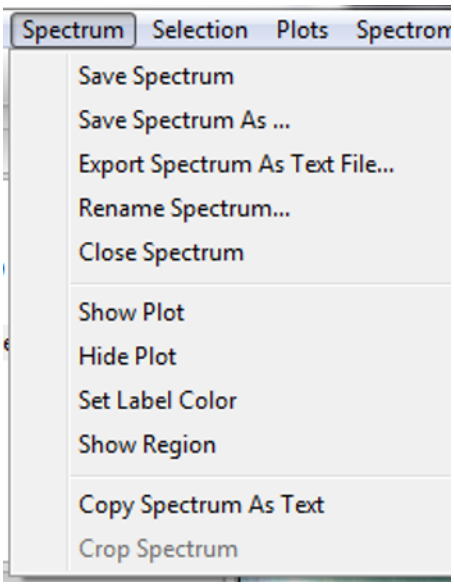
This tool will close the image shown in the current Image panel.

7.4 Spectrum Menu

The Spectrum menu provides tools for manipulating individual spectrum.

7.4.1 Save Spectrum

The **Save Spectrum** tool will save the spectrum selected in the **Resource Tree** in its current configuration. If the spectrum has not been previously saved, a window will open that allows you to rename the spectrum and browse to save it in the file of your choosing. Once saved, the new name will appear in the spectral plot panel. The color selected for the spectrum will be saved.



7.4.2 Save Spectrum As

This tool allows you to save a spectrum in a filename of your choice.

7.4.3 Export Spectrum as Text File

This tool allows the user to save a spectrum in a text file.

7.4.4 Rename Spectrum

This tool allows you to rename a spectrum.

7.4.5 Close Spectrum

This tool will close the spectrum selected in the **Resource Tree** and remove the plot from the spectral plotter.

7.4.6 Show Plot

This tool will plot the spectral curve in the spectral plotter selected in the **Resource Tree**. Typically this option is used after you have hidden the plot with the following **Hide Plot** option.

7.4.7 Hide Plot

This tool will hide the spectral plot shown in the spectral plotter. To use this tool, select the spectrum you wish to hide in the **Resource Tree**, and then click **Hide Plot**.

7.4.8 Set Label Color

This tool allows you to change the color of spectral curves as well as classification maps generated with the spectral curves. To change colors of spectral curves, select the spectral curve you wish to change in the **Resource Tree**. Clicking on **Set Label Color** will reveal a pallet of colors. Choose a new color and then click "OK" and the color will be reset.

7.4.9 Show Region

Use this tool to reveal the ROI region for the plot.

7.4.10 Copy Spectrum as Text

This tool allows you to easily transfer the spectral data to another program such as Excel or Notepad. Select the spectrum you wish to copy in the Resource Tree, and then click on **Copy Spectrum as Text**. This copies the spectral data onto your clipboard. You can now paste the data into a program of your choosing.

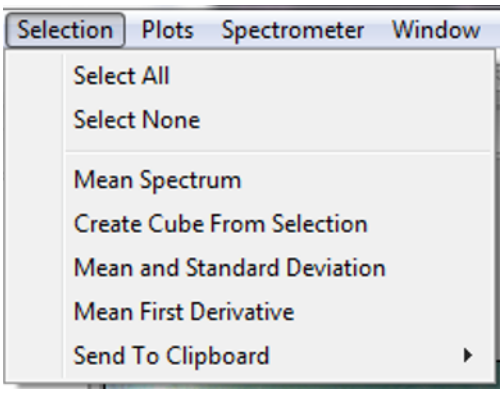
Hint: Positioning your mouse on a spectrum in the Resource Tree and right-clicking will open the Spectrum menu items described above.

7.4.11 Crop Spectrum

To use this tool, first select a spectrum, then select a region in the spectra window, and then click **Crop Spectrum**. The result will be a graph of the spectra in the region chosen. Note this tool can be accessed by right clicking the mouse after selecting a spectra.

7.5 Selection Menu

The **Selection** menu items provide a number of options for use with the selection tools (, , or). For all of the following options, you must first select a **Region of Interest (ROI)**.



Note: The **Selection** menu can be accessed from the main menu or by right-clicking in the image after creating an ROI.

7.5.1 Select All

This tool selects the entire dataset.

7.5.2 Select None

This tool clears the current selection.

7.5.3 Mean Spectrum

This tool will calculate and plot the **Mean Spectrum** in the spectral plotter for the pixels selected in your ROI. A plot color will be automatically assigned to the plot, and the spectrum will be listed in the **Resource Tree**.

7.5.4 Create Cube From Selection

This tool will generate and display a new datacube from the selected ROI. An image of your new datacube will appear in the Image panel and the new datacube will be listed in the **Resource Tree**. This tool allows you to easily crop your datacube into smaller pieces when you wish to concentrate on small regions of your original image, or to create complete new cubes of regions of interest for quantitative analysis. If the selected ROI is not rectangular, spectronon will create a rectangular representation of the pixels within the selected ROI. The broadcast of non-rectangular ROI to a rectangular shape may result in loss of spatial information, but all spectral information will be retained.

7.5.5 Mean and Standard Deviation

This tool is much like the **Mean Spectrum** option, except that it will also plot curves in the spectral plotter corresponding to the mean spectrum plus and minus the standard deviation of the brightness values within the ROI. This option enables easy visualization of the variability in your spectral data as a function of wavelength.

7.5.6 Mean First Derivative

This tool plots the first derivative of the mean spectral curve within the ROI as a function of wavelength in the spectral plotter, and also lists the curve in the **Resource Tree**.

7.5.7 Mean Second Derivative

This tool plots the second derivative of the mean spectral curve within the ROI as a function of wavelength in the spectral plotter, and also lists the curve in the **Resource Tree**.

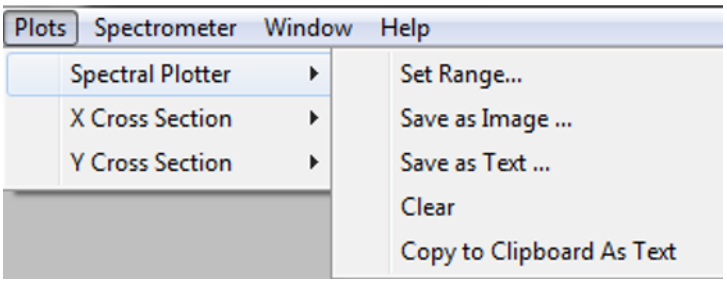
Hint: You may want to close all other spectra before using Mean First and Second Derivative tools, as the magnitude of the derivative values will typically be very different than the values of the spectra themselves.

7.5.8 Send to Clipboard

This tool allows you to transfer the data within your selected ROI. The first option is **Copy Mean as Text**, which allows you to paste the mean spectrum values into other applications such as Notepad or Excel. The second option, **All Spectra as Text** copies the spectral curves of all the pixels within your selected ROI. This tool is useful for those who want to perform specific statistical analysis on the data from small regions.

7.6 Plots Menu

The **Plots** menu items provide tools for the data shown in the spectral plotter.



7.6.1 Set Range

This allows the user to set the range of the X-axis and the Y-axis.

7.6.2 Save as Image

This tool allows you to save the plot as an image file. Clicking on this option will open a window that allows you to name the file, select the image format, and save the plot image file in a location of your choosing. This option is available for the **Spectral Plotter**, **X Cross Section** plot, and the **Y Cross Section** plot.

7.6.3 Save as Text

This tool allows you to save the plot data values as a .txt file. Clinking on this option will open a window where you can name the file for the .txt file and save it in allocation of your choosing. This option is available for the **Spectral Plotter**, **X Cross Section** plot, and the **Y Cross Section** plot.

7.6.4 Clear

This tool allows you to remove a spectral plot. To use this tool, select the Plot you wish to remove in the **Resource Tree**, then select **Plots** → **Spectral Plotter** → **Clear**

7.7 Datacubes and Header Files

Datacubes collected with Resonon imaging spectrometers are ENVI® compatible. ENVI is a standard software package for hyperspectral data analysis that can be obtained from ITT: http://www.ittvis.com/. ENVI compatible data requires two files, one that contains the hyperspectral data, and also a header file that contains important information about the data. Hyperspectral data can be accessed with Matlab; documentation for opening multi-band image files in Matlab can be found here: http://www.mathworks.com/help/techdoc/ref/multibandread.html

7.7.1 Hyperspectral data files

ENVI-compatible hyperspectral data can be stored in the following three formats:

Band-Sequential In Band-Sequential (BSQ) format each line of the data is followed by the next line in the same spectral band. Hyperspectral data stored in BSQ format will end with .bsq.

Band-Interleaved-by-Line Data saved in Band-Interleaved-by-Line (BIL) format have the first line of the first band followed by the first line of the second band, followed by the first line of the third band, and so forth. This pattern is repeated for subsequent lines. Hyperspectral data stored in BIL format will end with .bil.

Band-Interleaved-by-Pixel Band-Interleaved-by-Pixel (BIP) data have the first pixel for all bands in sequential order, followed by the second pixel for all bands, and so forth for all pixels. Hyperspectral data stored in BIP format will end with .bip.

Resonon imaging spectrometers will save hyperspectral data in BSQ, BIL, or BIP formats, and Spectronon software will open hyperspectral data in all of these formats.

7.7.2 Hyperspectral data header files

Header files for hyperspectral data have the same name as the BSQ, BIL, or BIP files followed by .hdr. The following discussion provides an overview of header files. For additional information, see for example: http://geol.hu/data/online_help/ENVI_Header_Format.html

One can open and edit header files using WordPad, for example. The following list describes some, but not all, of header file commands. Bold indicates the command, quantities in (brackets) are acceptable arguments for the commands, and statements in [square brackets] are additional explanation. Note that the order of these commands can be different from file to file.

ENVI [All ENVI-compatible header files begin with this]

interleave = (bil, bip, bsq) [Tells the software the hyperspectral data format]

data type = (4, 12) [4 is 4-byte floating point; 12 is 2-byte unsigned ; other data types are also supported, but these are common]

lines = (###) [The number of lines in the hyperspectral data where ### is a number]

samples = (###) [The number of samples in the hyperspectral data where ### is a number. This is typically the number of cross-track spatial pixels of the hyperspectral imager.]

bands = (###) [The number of bands in the hyperspectral data where ### is a number]

bit depth = (12, 14) [The bit depth of the hyperspectral data]

shutter = (###) [The shutter time in units of msec where ### is a number. This is mostly for reference, but it may be used by certain plugins]

gain = (#.#) [The camera gain in dB where #.# a number. This is mostly for reference, but it may be used by certain plugins]

framerate = (####) [The framerate at which the data were recorded in frames/sec where ### is a number. This is mostly for reference, but it may be used by certain plugins]

reflectance scale factor = (4095, 1023, 1) [If you divide the data by this factor, the data will scale from 0 to 1, typically for reflectance data]

byte order = (0) [byte order is the ordering of byte significance (http://en.wikipedia.org/wiki/Endianness) Data for Spectronon are LSF or Little Endian which means the least significant bytes come first. Spectronon will not properly open BSF or Big Endian files]

header offset = (0) [This is the number of bytes at the beginning of the binary file that should be ignored].

wavelengths = (###, ###, ###, . . .) [List of the spectral band centers in units of nm]

rotation = ((#,#), (#,#), (#,#), (#,#)) [This is a default orientation of the view of a datacube in Spectronon. This is a Spectronon only header option].

label = (name) [The label is used by Spectronon to give a human readable alternative to the file name of a file. If omitted, Spectronon will use the file name

timestamp = (day & time of datacube) [If available, this is when the datacube was recorded. The timestamp is often found on airborne datacubes, and datacubes where a script entered this information explicitly into the header]

The following are for spec.hdr files only, headers for spectrum. Spectrum files are ENVI compatible because they are 1x1 datacubes, but Spectronon adds in these data for its own use.

original cube file = (name of original cube) [This is a Spectronon-only header item for recording the history of a generated spectrum. In the case of a derived datacube, you will see a "history" header value that shows the way this cube was generated]

pointlist = (long list of values) [This is the set of points from the original cube that were averaged together to make this spectrum if the spectrum is a mean]

label color = (#FF00FF) [This is the color used to plot the spectrum]

CHAPTER

EIGHT

ADVANCED DATA ANALYSIS 2: HYPERSPECTRAL CLASSIFICATION

8.1 General Approach

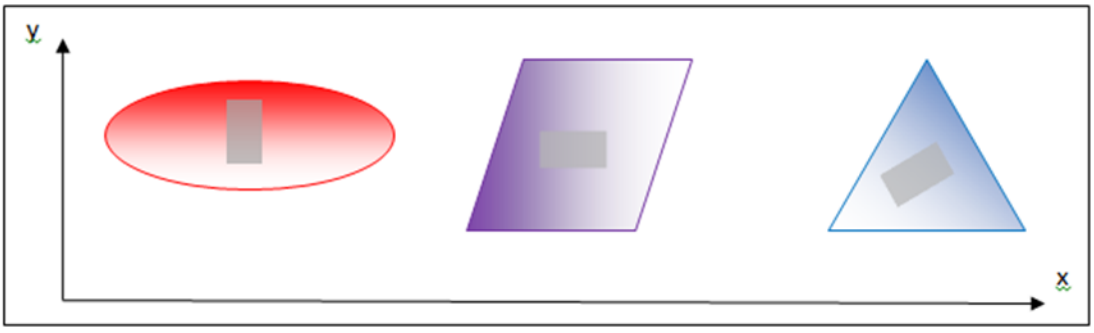
The detailed spectral information in hyperspectral data enables one to distinguish between very similar objects. Additionally, machine vision algorithms enable rapid, accurate, and repeatable classification of objects. With these capabilities, hyperspectral imaging has a broad range of current and potential applications, including sorting (food, raw materials, recycled materials), quality control (pharmaceuticals, food, printed goods), remote sensing (defense, search & rescue, mineral exploration, agriculture), to name just a few.

Critical to all of these applications is implementation of algorithms that classify the pixels within an image based on their spectral profiles. This chapter provides an introduction to hyperspectral data classification. There are many approaches to classifying objects. However, the general approach to most of these algorithms can be understood by considering the following simplified hypothetical examples.

In general, the algorithms used to classify objects scale with the number of spectral channels. Understanding how the general approaches work with a simple 2-color system makes it relatively easy to understand hyperspectral classification with tens or hundreds of channels. Therefore, consider a 2-color camera that provides digital numbers for how much red and how much blue is in each pixel. (Thus, this example is even easier than a conventional digital color camera that provides digital values for the three colors: red, green, and blue.)

Much like humans need to learn to distinguish between objects, classification algorithms generally also must be trained. Typically this is done by imaging samples of interest, and then using results from this "training set" to learn how to distinguish between objects in general.

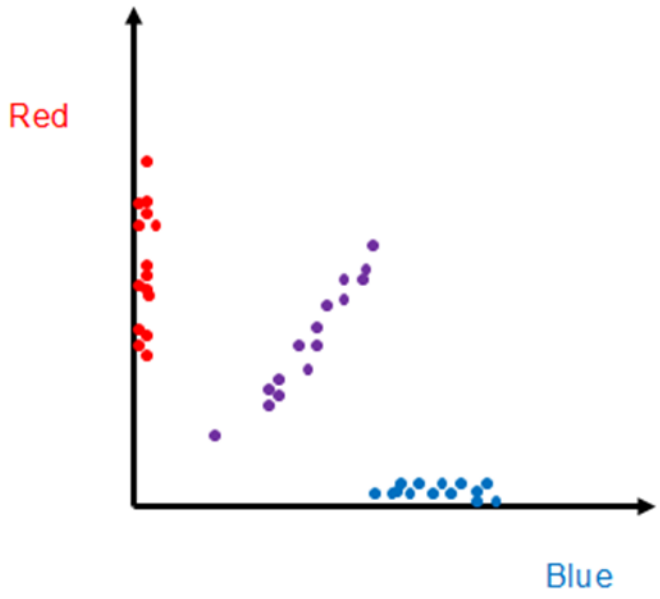
As an example, assume we wish to distinguish between the three objects shown below, a red ellipse, a blue triangle, and a purple (mixture of red and blue) parallelogram. (Ignore the grey rectangles for now.) An x and y axis is drawn to provide location coordinates for each pixel.



Note that each object has a distribution of light and dark pixels, although each object is approximately the same "color". If we image these objects with our hypothetical 2-color imager that senses only "red" and "blue" channels, there will be two spectral channels per pixel. To train our imaging system, we first select a representative set of training

pixels from the image of each of the objects of interest. This might be done, for example, by selecting the pixels within each grey rectangle indicated in the image above.

A useful way to visualize the color information in these training pixels is to plot the red and blue brightness values for each selected pixel in "color space" along blue and red axes, as shown below. (Note: This color plot uses only the training pixels selected within the three grey rectangles shown in the image above.)

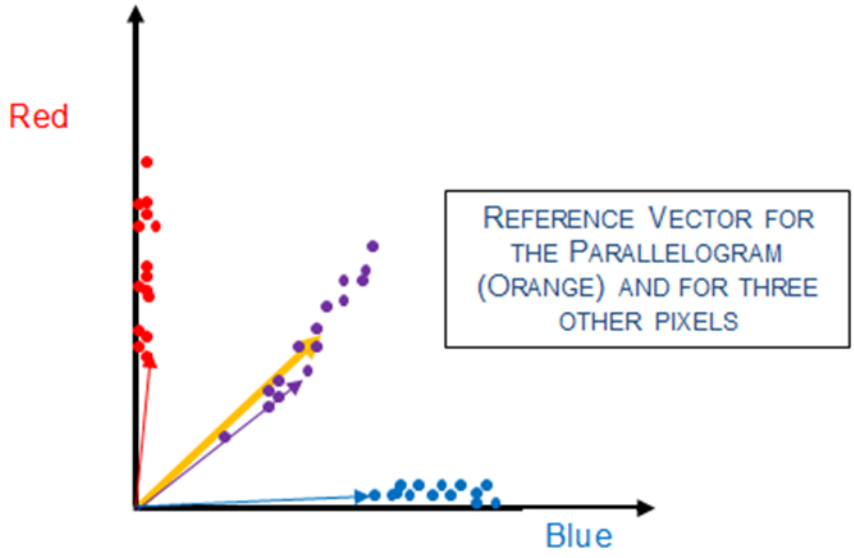


We can immediately recognize the training pixels that align along the vertical (red) axis are associated with the red ellipse, as those pixels clearly have large red brightness values, but small blue brightness. Similarly, the blue triangle pixels lie primarily along the horizontal (blue) axis. The purple pixels, however, lie in between the two axes because the color purple has significant red and blue brightness (i.e., it is a mixture or red and blue). Because some of the pixels are dark and others are light, the distribution of training pixels from each object is spread out from small values to large values in a near-linear manner.

One can see that the pixels from the spectrally distinct objects are separated in this "color space". Although this example has a 2-dimensional color space, one could create a 3-dimensional color space for RGB cameras, or 100 dimensions for 100-band hyperspectral imagers. In general, additional dimensions provide additional "opportunities" for points from different objects to be distinct in the color space, and thus they are easier to classify in practice, but otherwise the number of dimensions need not concern us now. Admittedly it is difficult to visualize a 100-dimension space, but the mathematical transition is often straight-forward. Consequently, one can invent techniques that work in two dimensions, and they can generally be applied to 100 dimensions.

Consider the following example of the well-known hyperspectral classification algorithm known as Spectral Angle Mapper (SAM). Note that the groups of training pixels from the three different objects in our 2-dimensional color plot are located at different angles relative to the horizontal axis. The SAM technique utilizes this property to classify ALL pixels.

Consider a reference vector at the center of the group of purple training pixels. This vector is shown as a large orange vector below. Similarly, we can envision vectors from the origin to each pixel in the plot (only three are shown to reduce clutter).



One can see that the angle between the reference orange vector is small for the purple pixels, and relatively large for the red and blue pixels. Thus, if we calculate the angle between the reference orange vector and all pixels in the image, we recognize that those pixels with a small angle are from the purple parallelogram, and those pixels with a large angle from the orange reference vector are not purple pixels.

Fortunately there is an easy way to calculate the angle between two vectors by utilizing the vector dot product. To do this, write the orange reference vector in component form as

$$\vec{R} = (R_r, R_b)$$

where the subscript r indicates the red brightness value component and b indicates the blue brightness value component. Similarly, for all pixels in the image, write

$$\vec{P}(x,y) = (P_r(x,y), P_b(x,y))$$

where x and y indicate the location of the pixel in the original image with an ellipse, triangle, and parallelogram. The vector dot product of $\vec{R}$ and any pixel $\vec{P}(x,y)$is

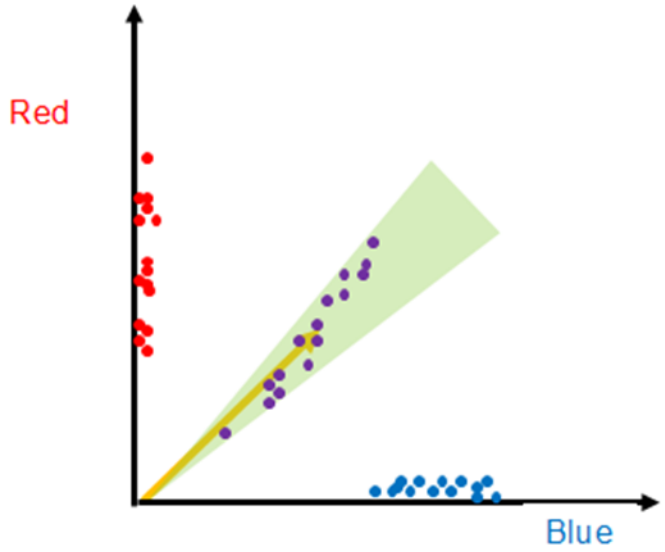
$$\vec{R} \bullet \vec{P}(x,y) = R_r P_r(x,y) + R_b P_b(x,y) = |\vec{R}||\vec{P}(x,y)|cos(\theta(x,y))$$

where $|\vec{R}|$and $|\vec{P}(x,y)|$ are the magnitudes of $\vec{R}$ and $\vec{P}(x,y)$. E.g., $|\vec{R}| = [R_r R_r + R_b R_b]^{1/2}$ and $\theta(x,y)$is the angle between the reference vector $\vec{R}$ and the pixel vector $\vec{P}(x,y)$.

Solving for $\theta(x,y)$yields

$$\theta(x,y) = cos^{-1}\left(\frac{\vec{R} \bullet \vec{P}(x,y)}{|\vec{R}||\vec{P}(x,y)|}\right)$$

By choosing only those pixels that have an angle θ (x,y) less than some small threshold value, one can effectively identify all the purple pixels. Graphically, this is equivalent to choosing only those pixels within a narrow green cone, as shown below.



By finding the orange reference vector and establishing the threshold of acceptable angles from this vector, we have "trained" our imaging system. To identify objects with the same color as the purple parallelogram, one images the objects and then calculates the angle as described above for each pixel. Pixels with an angle smaller than the threshold are identified to be the same material (color) as the purple parallelogram.

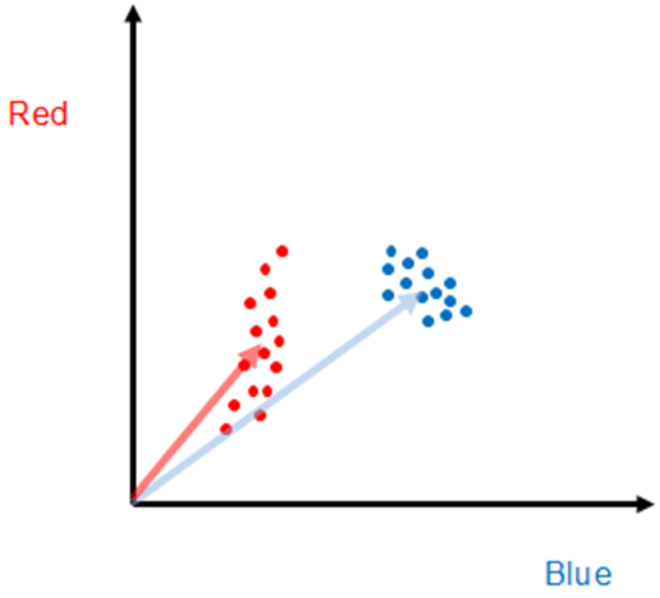
One could also train the system to find pixels the same color as the red ellipse or blue triangle by finding the appropriate reference vector for these objects, thereby enabling one to identify multiple objects within one image by calculating the angle between multiple object reference vectors.

To extend this approach to hyperspectral data with multiple spectral channels, one utilizes the more general definition of a vector dot product

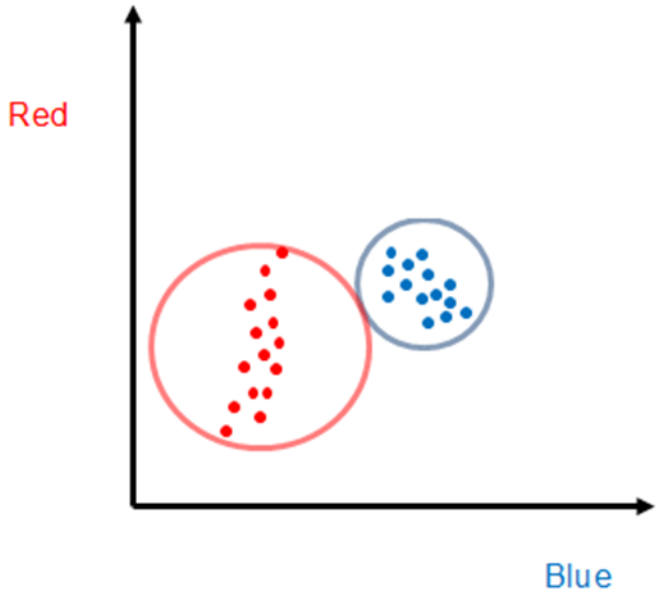
$$\vec{R} \bullet \vec{P}(x,y) = \sum_i R_i P_i(x,y) = |\vec{R}||\vec{P}(x,y)|cos(\theta(x,y))$$

where the sum is over all spectral bands (dimensions) indexed by i. The angle $\theta(x,y)$ is still the angle between the two vectors, $\vec{R}$ and $\vec{P}(x,y)$, although now in a complex multi-dimensional color space that is more difficult to visualize – the mathematics and approach are identical.

Of course SAM is not always the best algorithm to use, as one can see with the example below, where pixels from two hypothetical objects we wish to distinguish between are shown as red and blue points. In this case, if one choose representative vectors in the middle of each cluster to perform SAM classification, it is easy to see there would be substantial misclassification. (E.g., two red points lie nearly along the blue representative vector.)



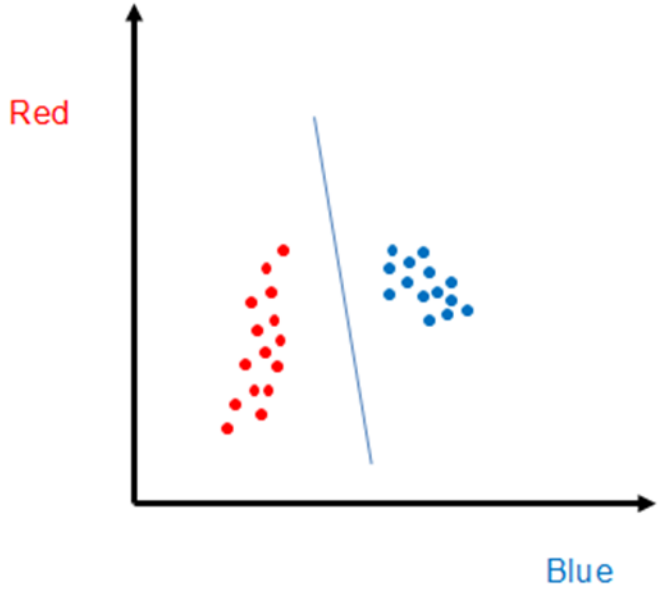
For the case shown above, a different algorithm is needed. One approach that would clearly be superior to SAM, at least for these data, is to find the center points of each set of training points, and then all points within a certain radius would be classified as that object. This is also a well-known hyperspectral classification algorithm known as Euclidean distance. Again, this approach is readily extended to hyperspectral data because the concept of distance between points is well established for multi-dimensional space.



The Euclidean distance approach can readily be improved upon by recognizing that a circle is not the best "enclosure", and an ellipse whose axes were scaled to the point distribution width would be far better. The hyperspectral classification algorithm that utilizes this information on the distribution of training points is called the Mahalanobis distance approach. As one would expect, the cost of additional algorithm sophistication is often the need for additional processing.

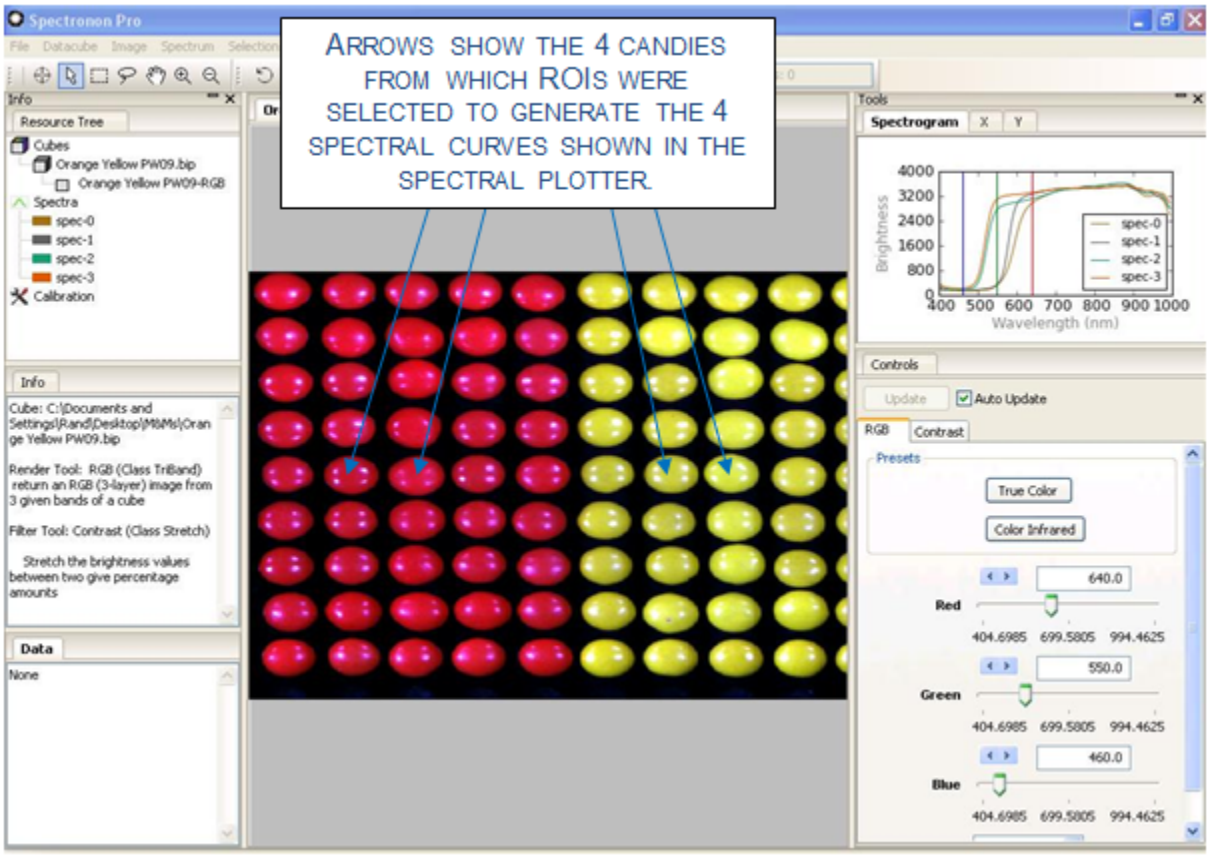
There are, of course, many other approaches to classifying objects using hyperspectral data, such as the one indicated

below where one draws a line between the two sets of training points. The system is trained by noting that all pixels that map to the left of the line are associated with one object, whereas the pixels that map to the right of the line are associated with the other. Extrapolating this approach to higher dimensions is more difficult, as the line becomes a plane in three dimensions, and a hyper-plane in color spaces with dimensions larger than three.

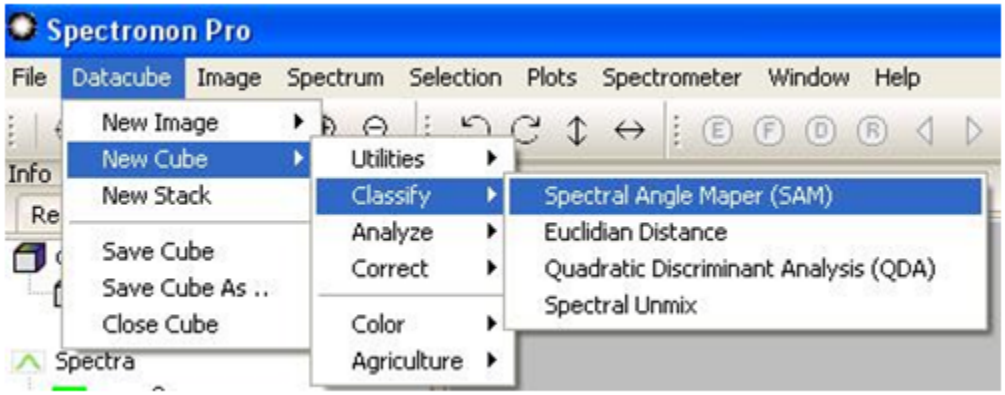


8.2 Spectral Angle Mapper (SAM) Classification

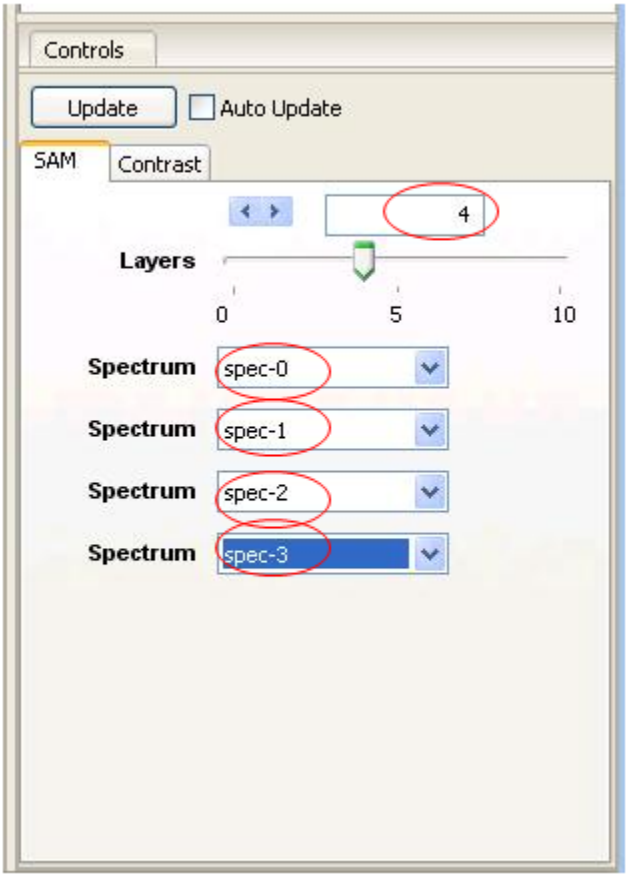
In this example, different kinds of M&M® and Reese's® Pieces candies are classified using the same hyperspectral datacube shown in Chapter 5. (This datacube can be downloaded from Resonon's website.) To perform SAM, reference spectra must be collected for the objects of interest. In this case, based on prior knowledge, we know that the four candies indicated below (see arrows in image) are all different. Using the marquee tool or lasso, select small ROIs on each of the four candies (avoid the glare spots) indicated in the figure below. After selecting the ROI, right-click, and then select Mean Spectrum. This will generate four spectral curves in the spectral plotter, and also list the four spectral curves in the Resource Tree, as shown below.



The four spectral curves shown in the spectral plotter will be used as the Reference Spectra to perform a SAM classification. To perform the classification, click on **Datacube** → **New Cube** → **Classify** → **Spectral Angle Mapper (SAM)**.



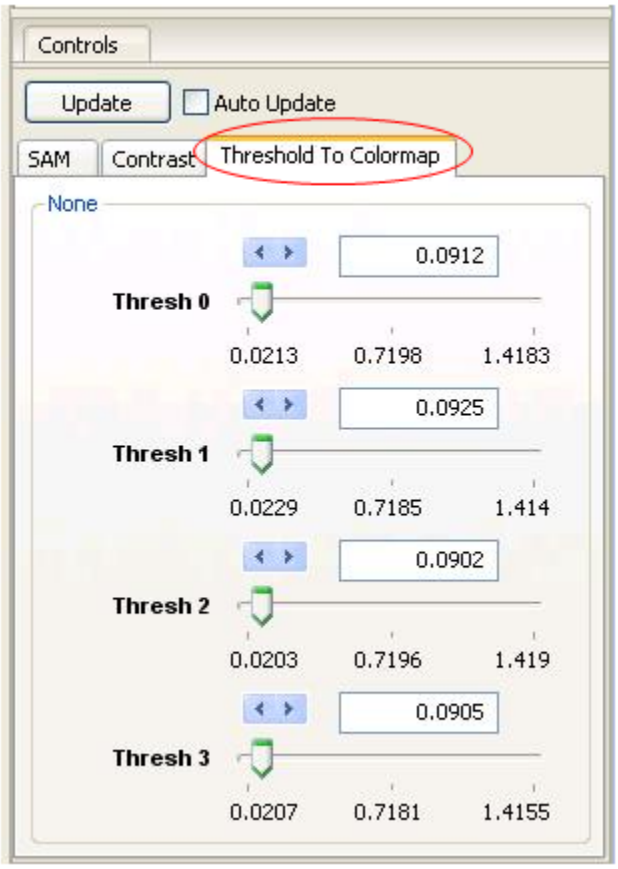
This will open a new window as shown. We wish to classify 4 objects, so use the slider or arrow keys to select a Spectrum for each box.



This will bring up 4 Spectrum pull-down menus. Click into each one of these and select one of the 4 spectra created with the ROI tool.

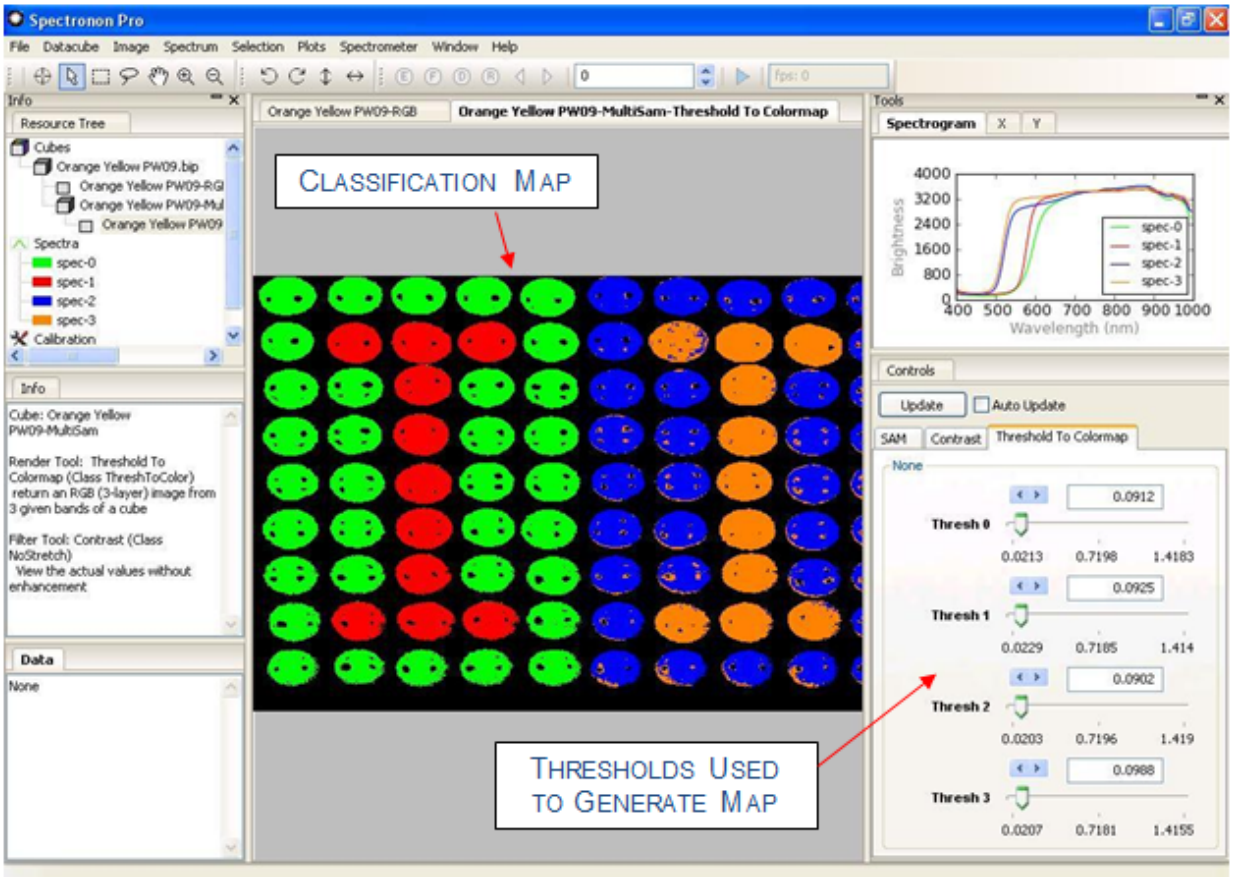
Then click **OK**, and Spectronon calculates the spectral angle (as described above) for each pixel for all 4 Reference spectra.

Typically, several seconds are required for the calculation, which generates a new classification map in the image panel using a default set of Threshold values.



Adjust the Threshold values to obtain a more accurate classification by clicking on the **Threshold to Colormap** tab in the tool control panel.

To adjust the Thresholds, move the sliders, select the arrow keys, or enter values by hand. Each Reference spectrum has its own threshold. After adjusting a threshold, click **Update** and a new classification map will be generated in the Image panel. With a few tries, a classification rendering similar to the one shown below can be generated. Only those pixels within the threshold are colored (the classification colors match those shown in the spectral plotter and **Resource Tree** for the Reference spectrum). If a pixel's Spectral Angle is within the threshold for more than one Reference spectrum, the quantity (Spectral Angle)/(Threshold Value) is calculated for each Threshold and classified for the class that minimizes this quantity. This weighting scheme allows you to emphasize or deemphasize each class to fine-tune your classification map.



SAM is only one of many, many possible classification algorithms. Other classification algorithms are accessed in a similar manner, as described in Chapter 8, section 2.2.2. These algorithms can be found by clicking *Datacube* → *New Cube* → *Classify*. Additionally, user-defined scripts can be written and used with Spectronon for custom classifications algorithms.

CHAPTER

NINE

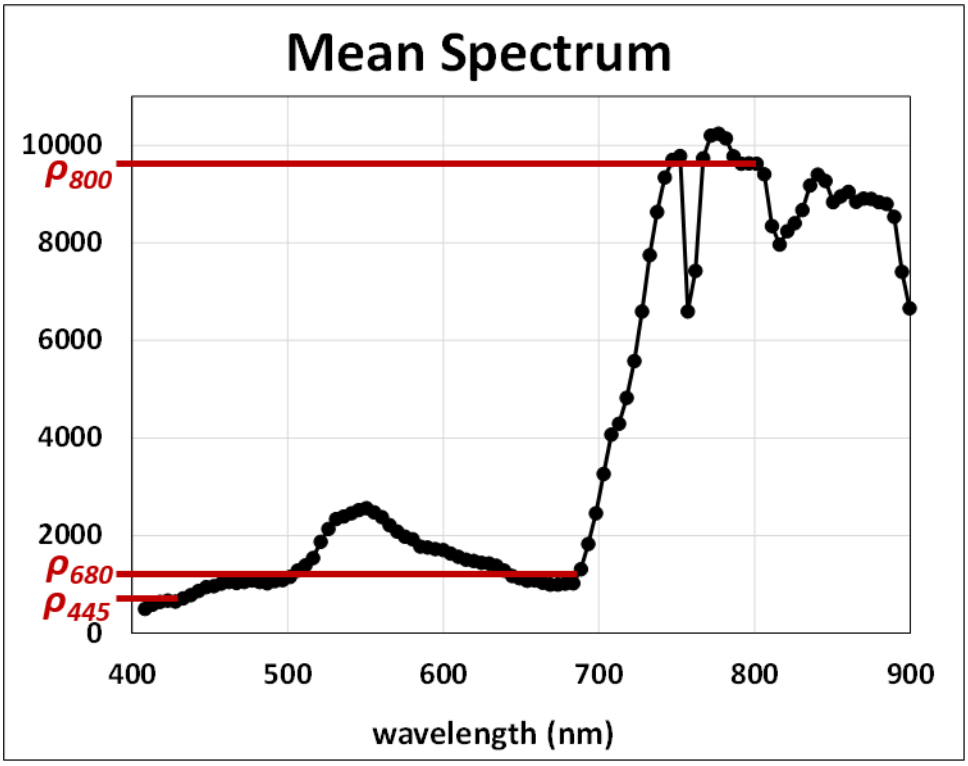
ADVANCED DATA ANALYSIS 3: HYPERSPECTRAL VEGETATION INDICES

9.1 Introduction

Hyperspectral Vegetation Indices (HVI's) are a common tool for analyzing hyperspectral data of agricultural crops. They are typically simple arithmetic combinations of sums and differences of spectral reflectance values at particular wavelengths. For example, the **Structure Insensitive Pigment Index** (SIPI) is defined by the following expression:

$$SIPI = \frac{\rho_{800} - \rho_{445}}{\rho_{800} + \rho_{680}}$$

where the values ρ_{445}, ρ_{680}, and ρ_{800} are shown graphically in the figure below:



HVI's have found many applications in research using airborne remote sensing data, and are currently subjects of intense research efforts.

9.1.1 Useful References

The developers at **ENVI** have written an excellent description of HVI's, which is available at their website. We at Resonon consider ENVI (owned by Exelis) to be excellent hyperspectral data analysis software. Here is a link to their HVI website:

https://www.exelisvis.com/Learn/WhitepapersDetail/TabId/802/ArtMID/2627/ArticleID/13742/Vegetation-Analysis-Using-Vegetation-Indices-in-ENVI.aspx

Another very good reference discussing HVI's is this book:

Thenkabail, Lyon, and Hute, *Remote Sensing of Vegetation*, CRC Press (2011).

9.1.2 HVI's in Spectronon

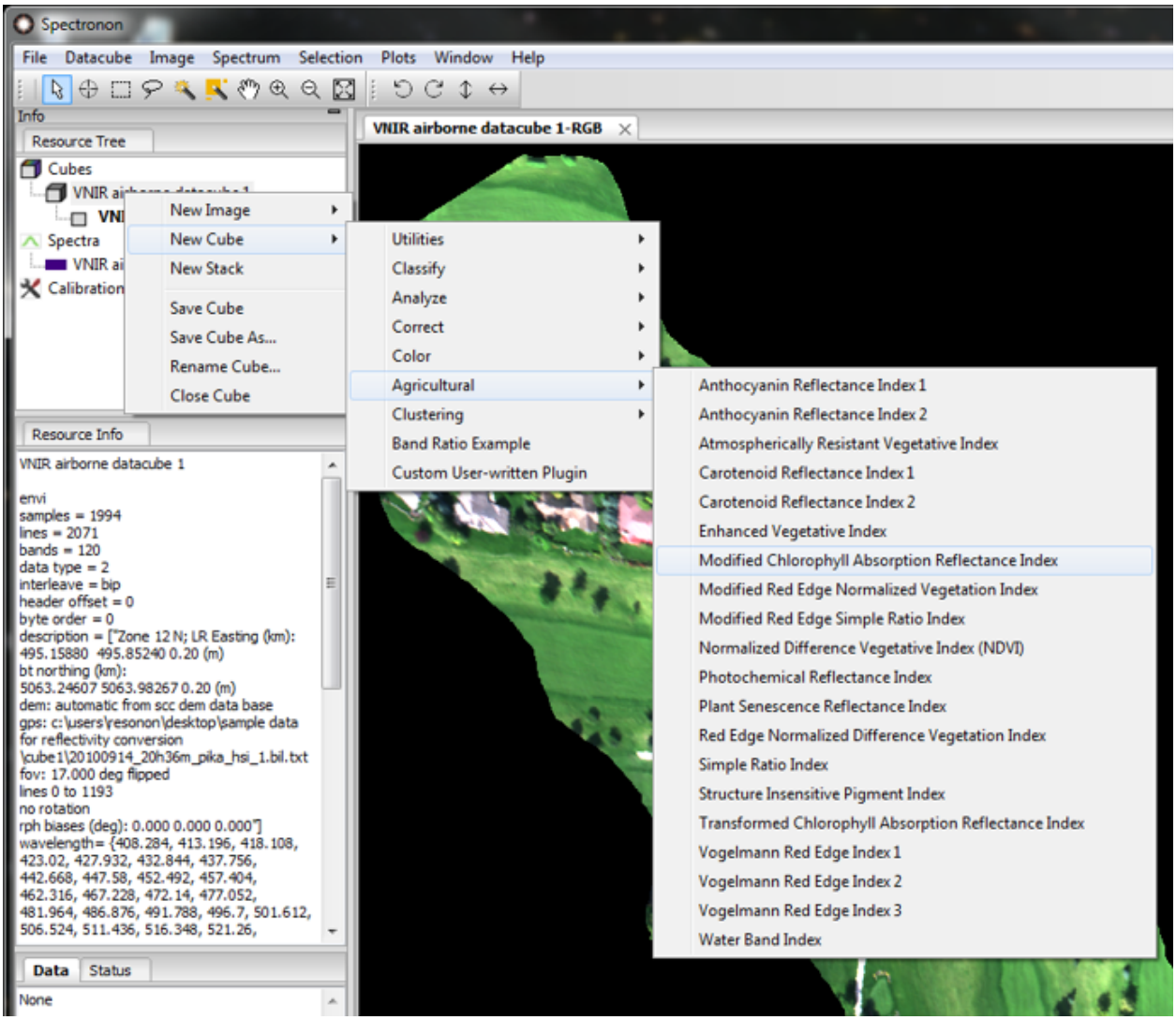
HVI's calculated by Spectronon are listed in the following table. Also listed are the equations that Spectronon uses for each HVI.

Hyperspectral Vegetation Index	Equation in Spectronon
Anthocyanin Reflectance Index 1	$ARI1 = \frac{1}{\rho_{550}} - \frac{1}{\rho_{700}}$
Anthocyanin Reflectance Index 2	$ARI2 = \rho_{800}\left(\frac{1}{\rho_{550}} - \frac{1}{\rho_{700}}\right)$
Atmospherically Resistant Vegatative Index	$ARVI = \frac{NIR - \left(Red - \gamma(Blue - Red)\right)}{NIR + \left(Red - \gamma(Blue - Red)\right)}$
Carotenoid Reflectance Index 1	$CRI1 = \frac{1}{\rho_{510}} - \frac{1}{\rho_{550}}$
Carotenoid Reflectance Index 2	$CRI2 = \frac{1}{\rho_{510}} - \frac{1}{\rho_{700}}$
Enhanced Vegative Index	$EVI = \frac{NIR - Red}{NIR + 6.0 \cdot Red - 7.5 \cdot Blue + 1}$
Modified Chlorophyll Absorption Reflectance Index	$MCARI = \rho_{700} - \rho_{670} - 0.2(\rho_{700} - \rho_{550})\left(\frac{\rho_{700}}{\rho_{670}}\right)$
Modified Red Edge Normalized Vegetation Index	$MRENDVI = \frac{\rho_{750} - \rho_{705}}{\rho_{750} + \rho_{705} - 2 \cdot \rho_{445}}$
Modified Red Edge Simple Ratio Index	$MRESR = \frac{\rho_{750} - \rho_{445}}{\rho_{705} - \rho_{445}}$
Normalized Difference Vegetative Index	$NDVI = \frac{NIR - Red}{NIR + Red}$
Photochemical Reflectance Index	$PRI = \frac{\rho_{531} - \rho_{570}}{\rho_{531} - \rho_{570}}$
Plant Senescence Reflectance Index	$PSRI = \frac{\rho_{680} - \rho_{500}}{\rho_{750}}$
Red Edge Normalized Difference Vegetation Index	$RENDVI = \frac{\rho_{750} - \rho_{705}}{\rho_{750} + \rho_{705}}$
Simple Ratio Index	$SR = \frac{NIR}{Red}$
Structure Insensitive Pigment Index	$SIPI = \frac{\rho_{800} - \rho_{445}}{\rho_{800} + \rho_{680}}$
Transformed Chlorophyll Absorption Reflectance Index	$TCARI = 3\left[\rho_{700} - \rho_{670} - 0.2(\rho_{700} - \rho_{550})\left(\frac{\rho_{700}}{\rho_{670}}\right)\right]$
Vogelmann Red Edge Index 1	$VREI1 = \frac{\rho_{740}}{\rho_{720}}$
Vogelmann Red Edge Index 2	$VREI2 = \frac{\rho_{734} - \rho_{747}}{\rho_{715} - \rho_{726}}$
Vogelmann Red Edge Index 3	$VREI3 = \frac{\rho_{734} - \rho_{747}}{\rho_{715} + \rho_{720}}$
Water Band Index	$WBI = \frac{\rho_{970}}{\rho_{900}}$

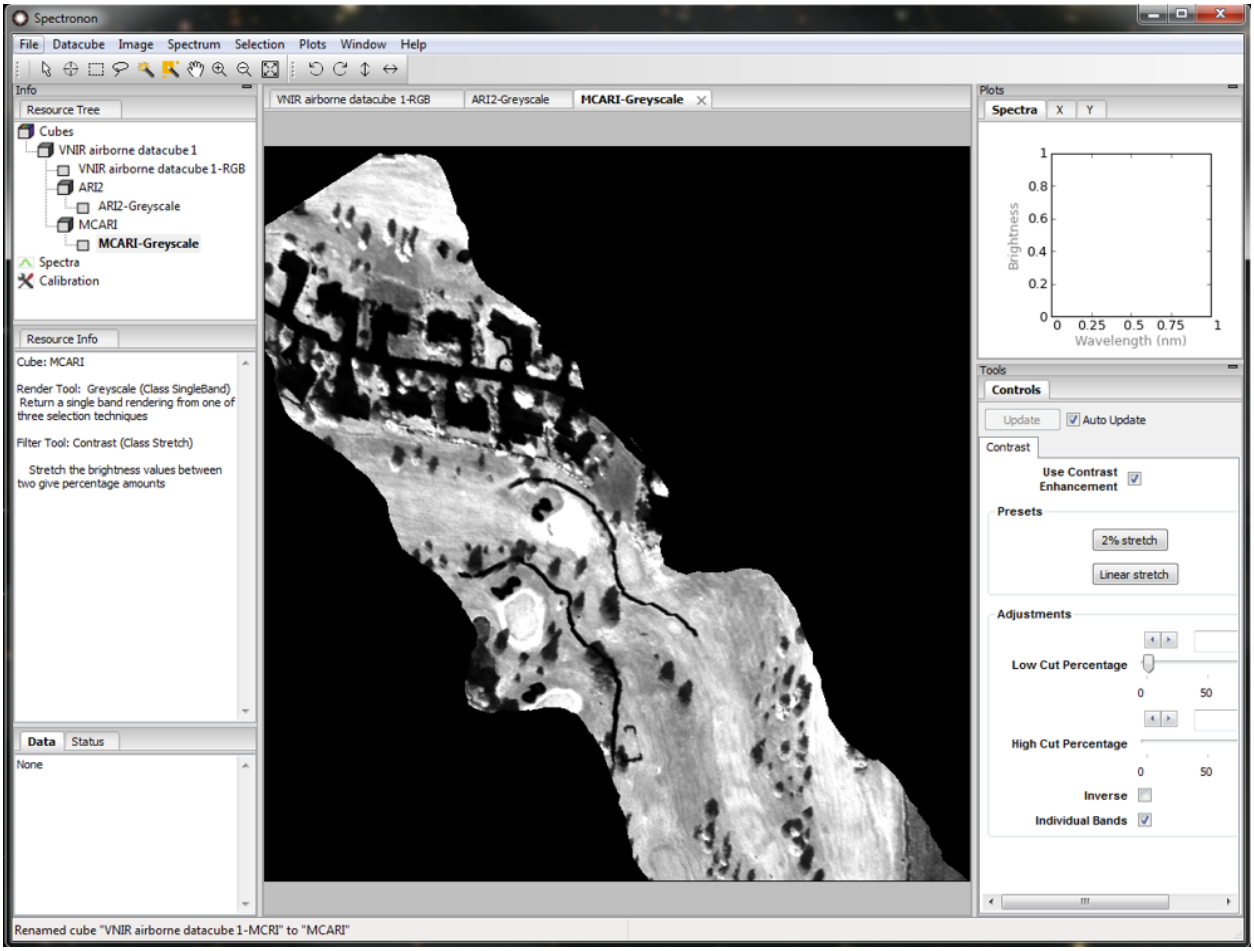
9.2 Generating HVI maps in Spectronon

The datacube in the following example was acquired using a Resonon airborne hyperspectral imaging system. This datacube is available for free download from http://downloads.resonon.com/.

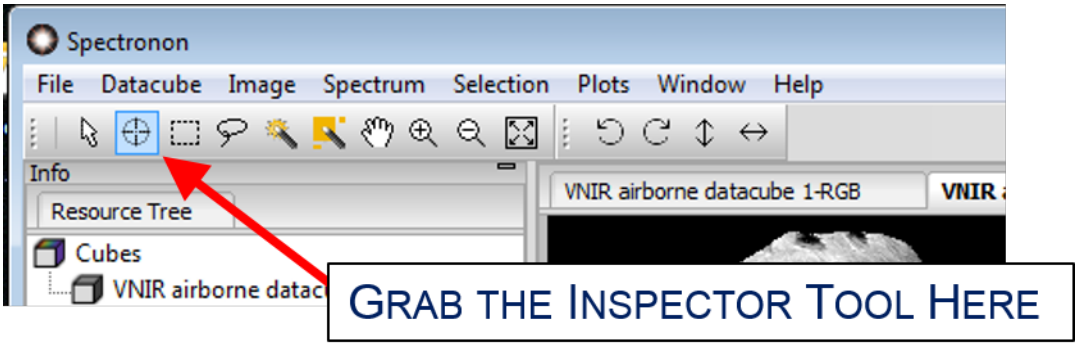
HVI's in Spectronon can be found at the path: **Datacube** → **New Cube** → **Agriculture**:



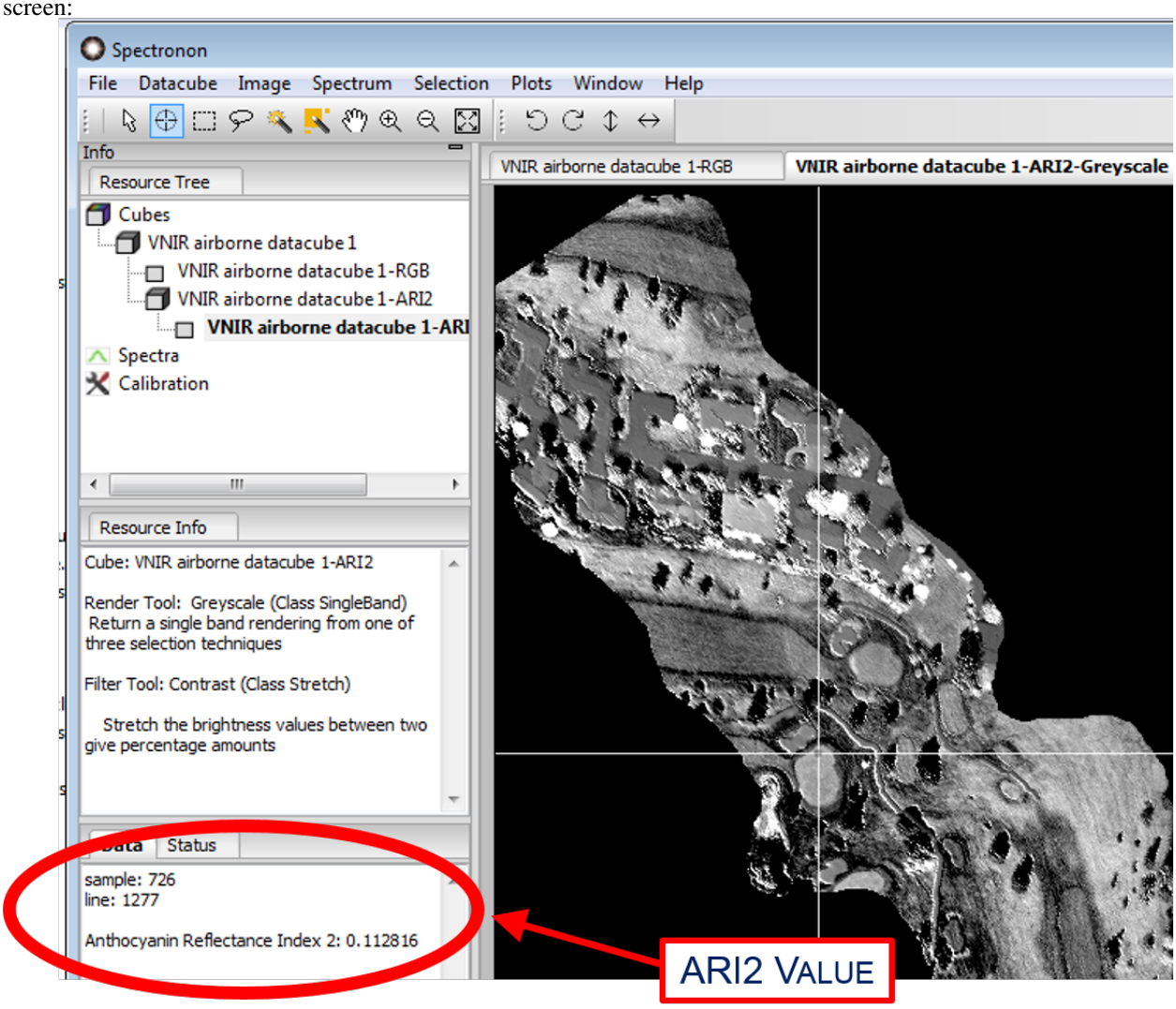
Choosing an HVI from this menu will generate a grey-scale image:



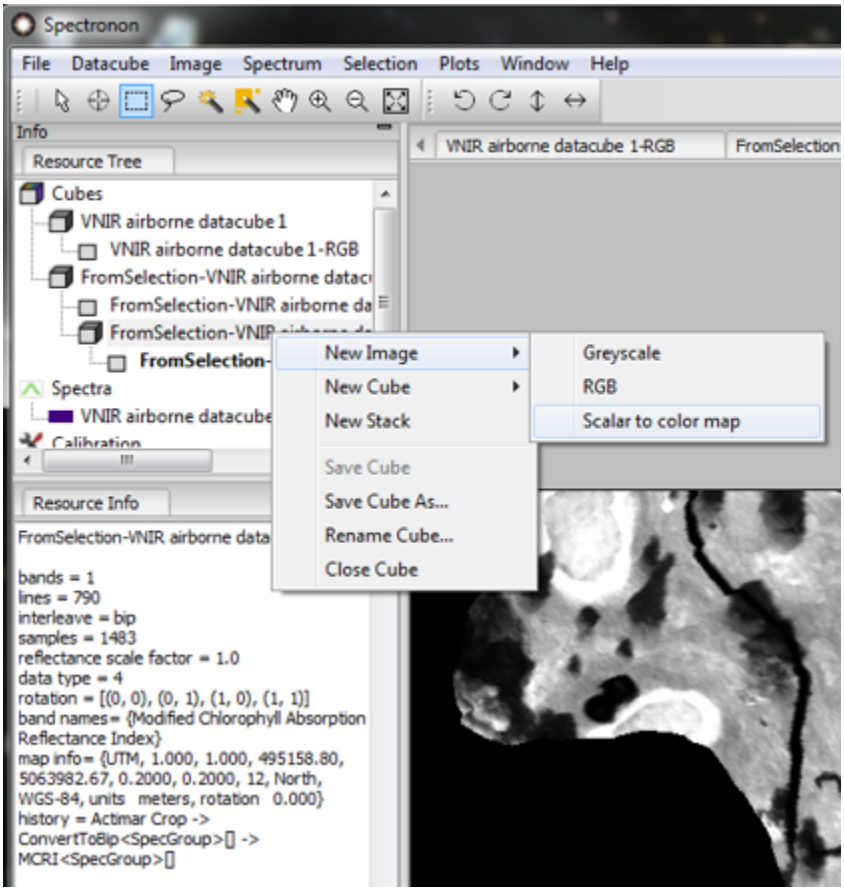
The value of the HVI at any pixel can be obtained with the Inspector Tool :



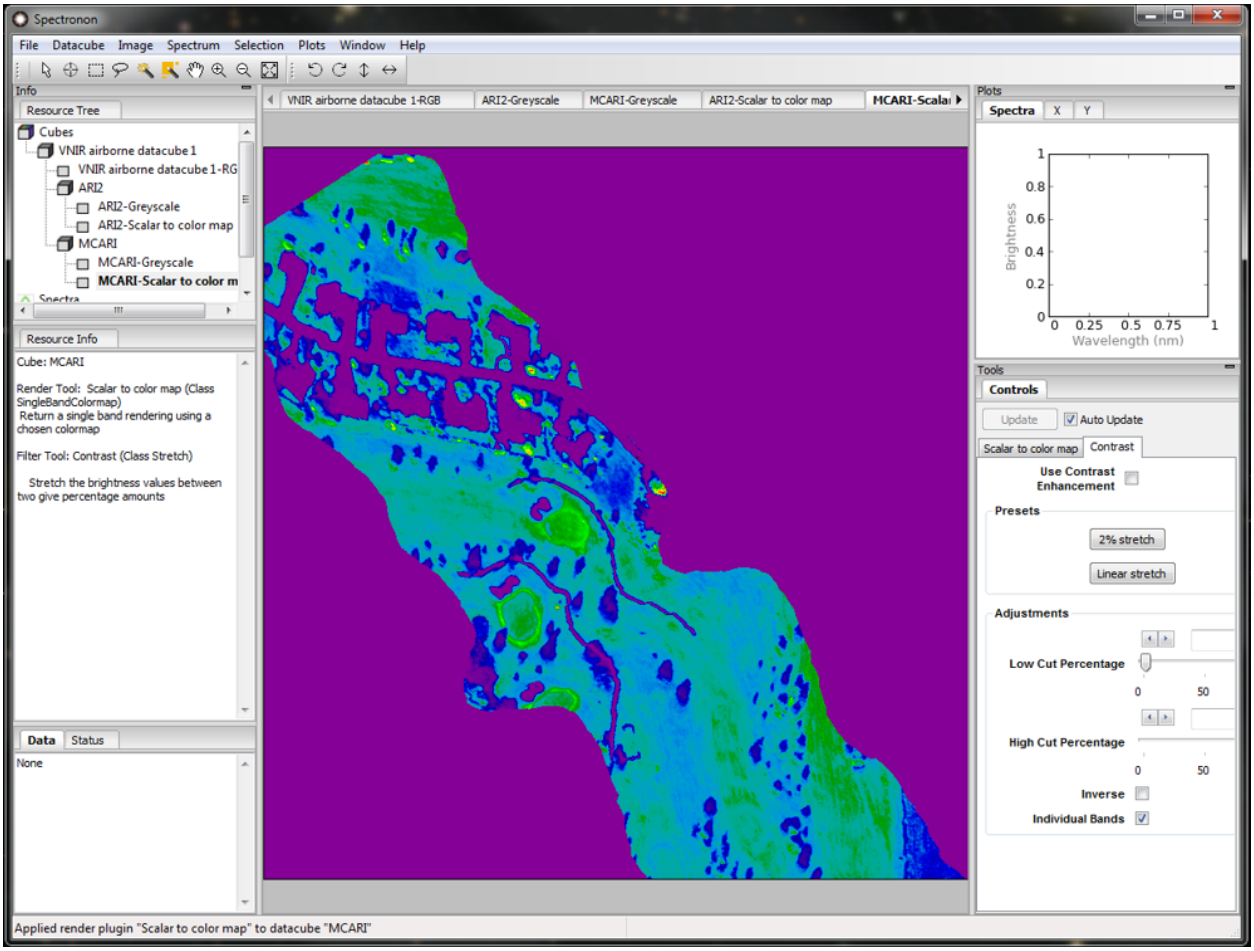
The value of the HVI, along with the coordinates of the inspector tool, can be found in the lower left corner of the screen:



Another tool for visualizing HVI maps is to transform the grey-scale image into a colormap. This can be accomplished with **Datacube** → **New Image** → **Scalar to Colormap**:

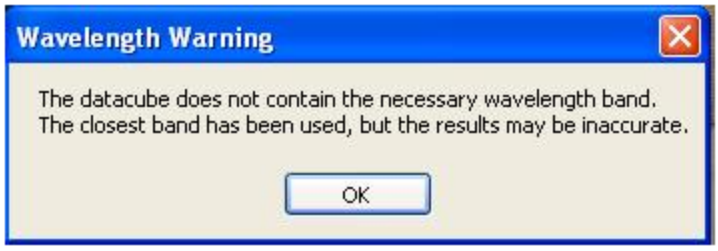


Below is a “spectral” colormap of the MCARI index for the example datacube:



After generating a colormap, the user can change the color scheme from the **Colormap** menu on the **Controls** tab. It may be helpful to toggle the **Contrast Enhancement** feature. Experiment with different colormaps, normalization, and contrast settings, all available on the **Controls** tab.

Note: If an HVI is selected that requires spectral bands that do not exist in your datacube, the warning shown below will appear. As described in the warning, clicking "OK" will generate an image using the closest bands available, but the resulting image will often be a poor approximation to a true index mapping.


Wavelength Warning
The datacube does not contain the necessary wavelength band.
The closest band has been used, but the results may be inaccurate.
OK

CHAPTER

TEN

CUSTOM DATA ANALYSIS: WRITING PLUGINS

Spectronon has the capability to support user-defined data analysis algorithms, commonly called **plugins**. Writing plugins allows the user to extend Spectronon's functionality by incorporating custom tools designed for their specific applications. Furthermore plugins can create interactive user interfaces and integrate directly into Spectronon's menu system.

Warning: Plugins have direct access to much of Spectronon's internal code. We make this functionality available in the hope that it will be a useful tool. Use of this tool is at the user's own risk. Resonon cannot support every situation that may arise from the use of this information. The methods and objects documented here may change at any time. Only the methods and attributes documented here are intended to be used in custom plugin development – all other methods and properties of the workbench, plugin, datacube, and spec classes are for internal use only and should not be called or overridden.

10.1 Getting started

Spectronon plugins are written in the Python programming language, and a working knowledge of Python is a prerequisite for writing Spectronon plugins. If you're new to programming or to Python, the Beginner's Guide to Python provides a collection of useful resources for getting started. Additionally, familiarity with the computing tools package Numpy is important, as Resonon datacubes are based on Numpy arrays. Finally, the SciPy library provides an excellent collection of tools for scientific computing in Python using Numpy arrays. Many Spectronon features make use of SciPy libraries, and SciPy is available to the user for use in developing additional plugins.

To develop a Spectronon plugin, create a file with a ".py" extension (such as MyCustomPlugin.py), and place it in the folder:

```
SpectrononAppData\user_plugins\
```

This folder is typically located at

```
C:\Users\<your user name>\SpectrononAppData\user_plugins\.
```

In the file write a subclass of the plugin type you desire (as described below). You can use any text editor of your choice to create the plugin - a list of Python text editors can be found here. Upon loading, Spectronon will scan the user plugins floder for .py files and, if a valid plugin is found, add it to the Spectronon menu system.

10.2 Spectronon Plugin Overview

There are 4 types of plugins:

- CubePlugin
- RenderPlugin

- SelectPlugin
- FilterPlugin (built-in plugins only)

Plugins are creating by defining a subclass of one of the base plugin types, then placing the source file that defines that plugin in the `spectrononAppData/user_plugins` folder. The base plugin type definitions must be imported from the `spectronon.workbench.plugin` package and subclassed. Each plugin class has three methods that you can override to define your custom behavior. For example:

```
from spectronon.workbench.plugin import CubePlugin

class MyPlugin(CubePlugin):
    """
    The docstring text will appear on the user interface as help text when hovering
    over the plugin's menu item.
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        """
        The setup method is where you will create the plugin's user interface.
        You can also define any custom initialization behavior here.  Do not override
        CubePlugin.__init__().
        """

    def update(self):
        """
        The update method is called each time the user changes the value of a Spec.
        You might need this for your plugin's book keeping.
        """

    def action(self):
        """
        The action method is called when the plugin is run, and should contain the
        logic of your custom behavior.
        """
```

Always define a label attribute as above, which will become the text that is displayed in Spectronon menus. The docstring used for the plugin class will become the help text when the user hovers over your custom plugin's menu item.

Always define `userLevel = 1` at the top of the class definition. Occasionally a user may use user level 2, which will render the plugin invisible unless Spectronon is invoked in expert-user mode.

Warning: Failure to set the userLevel to 1 or 2 at the top of the plugin definition will prevent the plugin from attaching to Spectronon's menu system.

Writing any plugin requires at minimum that you write an action method.

```
def action(self):
    # your stuff here
```

The action method is called by Spectronon when the plugin is activated, and again each time the user interacts with any Specs defined by the plugin. This method returns the results of your plugin.

- Cube plugins must return a datacube from their action method.
- Render plugins return an image representation of a datacube as a numpy array.

- Select plugins do not return a value, but interact directly with the workbench.
- Filter plugins cannot be written by users.

If you want to get information from the user, you also define a 'setup' method and define some *Specs* (more on specs below) within it.

```
def setup(self):
    # define some specs here
```

If your UI can be altered based on user entry into other UI elements (for example, the user selects how many inputs they want to provide), then you must write an 'update' method.

```
def update(self):
    # add or remove specs or update paramaters of existing specs
```

None of these methods accepts arguments. Instead, there are attributes of the Plugin that you can use.

10.3 The attributes available to each plugin type

`self.wb` (All Plugins) This is a handle on the spectronon workbench. The workbench gives you a lot of freedom, so use this reference wisely. Some common uses of self.wb are:

- popup dialog messages such as: `self.wb.postMessage("The combination of arguments is invalid for this cube")`
- retrieve data from the workbench: `correctioncube = self.wb.getCube(self.correctCubeID.value)`
- plot something to the plotter: `self.wb.plot(myarray)`
- and there are specs (SpecCube, SpecSpectrum) that expect the workbench as a parameter to help them work their magic

For more information, see *Using the Workbench*

`self.datacube` (CubePlugin, RenderPlugin, SelectPlugin)

This is the datacube on which this plugin will be applied. Cube plugins action methods are expected to return a new datacube based somehow on this one. Render plugins will use this datacube as a data source to build a viewable image.

For more information, see *Using Datacubes*

`self.pointlist` (SelectPlugin)

For any selection this is a list of points that was inside the selected area in the form of a numpy array of shape (number of points, 2) where self.pointlist[i,:] = (sample, line) index of the ith selected point.

10.4 Specs

A Spec is a class that manages a piece of data and a user interface to that data. They underly much of Spectronon's codebase. A Spec maintains the connection between the user interface and the data in the back end. When a plugin class has a spec member, Spectronon will generate an appropriate graphical widget to allow the user to adjust that spec's value in the plugin's control panel. Specs can be imported from `resonon.utils.spec`. Usually you will access the spec by getting the current value in an action method using the spec's 'value' member.

```
value = mySpec.value
```

Here is the list of available Specs:

General purpose input:

- *SpecBool* - allows the user to specify a true of false value
- *SpecFloat* - allows the user to specify a floating point value within a defined range
- *SpecInt* - allows the user to specify an integer value within a defined range
- *SpecChoice* - allows the user to choose an item from a defined list

Selecting references to cubes and spectra on the workbench:

- *SpecCube* - allows the user to select one of the datacubes currently available on the workbench
- *SpecSpectrum* - allows the user to select one of the spectra currently available on the workbench

Selecting portions of cubes or specta:

- *SpecWavelength* - a spec that accepts a datacube as an argument and allows the user to select a wavelength from that cube
- *SpecBandNumber* - a spec that accepts a datacube as an argument and allows the user to select a single band from that cube.

All specs have certain members that can be set to impact the way the spec appears and behaves:

`self.label`: The label that will appear on the graphical user interface next to the Spec's widget.

`self.units`: A string describing the Spec's units that will appear on the GUI if not set to None.

`self.help`: A string that will appear if the user hovers the mouse over the Spec.

10.4.1 SpecBool

```
class SpecBool(Spec):
    """
    Creates boutton or a checkbox for entering a boolean.
    """
    def __init__(self, label, defaultValue=True):
```

By default, a SpecBool is represented on screen by a checkbox. To change the interface type, set the `interfaceType` member of the spec. Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecBool
from resonon.constants import INTERFACES

class MyPlugin(CubePlugin):
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        self.my_bool = SpecBool(label='My Option', default=True)
        self.my_bool.units = 'my units'
        self.my_bool.help = 'my mouse hover help'
        self.my_bool.interfaceType = INTERFACES.CHECKBOX

    def action(self):
        if self.my_bool.value:
```

```
            pass
                #action if true
        else:
            pass
            #action if false
```

10.4.2 SpecFloat

```
class SpecFloat(SpecNumber):
    """Floa
    Creates a slider or spinner for setting a floating point number
    """
    def __init__(self, label, minval, maxval, stepsize=1, defaultValue=None):
```

The default interface type of a SpecFloat is a slider. A spin button can also be used. Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecFloat
from resonon.constants import INTERFACES

class MyPlugin(CubePlugin):
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
         self.my_float = SpecFloat(label='My Float', minval=0, maxval=100,
                                   stepsize=0.1, defaultValue=10.4)
         self.my_float.units = 'my units'
         self.my_float.help = 'my mouse hover help'

         # INTERFACES.SPIN is also valid.  This line is not needed if you
         # just want default behaviour.
         self.my_float.interfaceType = INTERFACES.SLIDER

    def action(self):
         my_product = self.my_float.value * 2.5
         # more plugin logic here....
```

10.4.3 SpecInt

```
class SpecInt(SpecNumber):
    """
    Creates a slider or spinner for setting an int
    """
    def __init__(self, label, minval, maxval, stepsize=1, defaultValue=None):
```

Using a SpecInt is just like using a SpecFloat. See *SpecFloat* for an example.

10.4.4 SpecChoice

```
class SpecChoice(Spec):
    """
    Creates a combo box depending allowing selection of a value from a list of choices
    """
    def __init__(self, label, values=None, defaultValue=None):
```

A Spec choice allows the user to choose an option from a combo box. It expects a list of strings as possible choices. It's `value` member will be one of the strings in the list. Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecChoice

class MyPlugin(CubePlugin):
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        self.my_options = SpecChoice(label='option',
                                     values = ['choice one',
                                               'choice two',
                                               'choice three'],
                                     defaultValue = 'choice one')
        self.my_options.help = 'my mouse hover help'

    def action(self):
        if self.my_options.value == 'choice one':
            pass # action for choice one
        elif self.my_options.value == 'choice two':
            pass # action for choice two
        elif self.my_options.value == 'choice three':
            pass # action for choice three
```

10.4.5 SpecCube

```
class SpecCube(Spec):
    """
    Displays a ComboBox for selecting one of the loaded datacubes
    """
    def __init__(self,
                 label,
                 datacube,
                 wb,
                 requireMatchedLineCount=False,
                 requireMatchedSampleCount=False,
                 requireMatchedBandCount=False,
                 requireBandCount=None,
                 defaultValue=None):
```

The SpecCube user interface is a combo box that will automatically be populated with the datacubes that are currently available on the Spectronon workbench (e.g. any cube that appears in the Spectronon resource tree). The parameters passed to SpecCube's constructor allow you to provide criteria by which the list of applicable cubes will be filtered.

- if `requireMatchedLineCount=True` the list will only show cubes whose line count is equal to that of the cube passed as `datacube`.
- if `requireMatchedSampleCount=True` the list will only show cubes whose sample count is equal to that of the cube passed as `datacube`.
- if `requireMatchedBandCount=True` the list will only show cubes whose band count is equal to that of the cube passed as `datacube`.
- if `requireBandCount=[an integer]` the list will only show cubes whose band count is equal the specified integer. This value overrides that provided in `requireMatchedBandCount`.

Most commonly, you will pass your plugin's datacube member to SpecCube's constructor as the datacube argument, but any datacube object is a valid choice. You should always pass the plugin's self.wb (workbench) member to SpecCub as the wb argument.

The value of a SpecCube is an **index** to a cube within the workbench. To get the cube object itself, you must call the `wb.tree.getCube` method.

Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecCube

class MyPlugin(CubePlugin):
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        self.my_cube = SpecCube(label='Select a Datacube',
                                datacube=self.datacube,
                                wb=self.wb,
                                requireMatchedLineCount=False,
                                requireMatchedSampleCount=False,

                                # list only those cubes with the same number of
                                # bands as the cube we operate on
                                requireMatchedBandCount=True,
                                requireBandCount=None,
                                defaultValue=None)
        self.my_cube.help = 'my mouse hover help'

    def action(self):
        primary_cube = self.datacube
        secondary_cube = self.wb.tree.getCube(self.my_cube.value)
        # operate on the datacubes...
```

10.4.6 SpecSpectrum

```
class SpecSpectrum(Spec):
    """
    Displays a ComboBox for selecting one of the loaded spectra
    """
    def __init__(self,
                 label,
                 datacube,
```

```
                wb,
                requireMatchedWavelengths=False,
                requireMatchedBandCount=False,
                defaultValue=None):
```

SpecSpectrum is similar to SpecCube, but allows for choice of a spectrum object from the workbench.

- if `requireMatchedWavelengths=True`, the list will only show spectra for which the wavelengths of the spectrum exactly match those of the cube passed as `datacube`.
- if `requireMatchedBandCount=True`, the list will only show spectra for which the number of bands matches the number of bands of the cube passed as `datacube`.

Most commonly, you will pass your plugin's datacube member to SpecCube's constructor as the datacube argument, but any datacube object is a valid choice. You should always pass the plugin's self.wb (workbench) member to SpecCub as the wb argument.

The value of a SpecSpectrum is an **index** to a spectrum object within the workbench. To get the spectrum object itself, you must call the `wb.tree.getSpectrum` method.

Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecSpectrum

class MyPlugin(CubePlugin):
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        self.my_spectrum = SpecSpectrum(label='Select a Spectrum',
                                        datacube=self.datacube,
                                        wb=self.wb,
                                        requireMatchedWavelengths=True,
                                        requireMatchedBandCount=False,
                                        defaultValue=None)
        self.my_spectrum.help = 'my mouse hover help'

    def action(self):
        spectrum = self.wb.tree.getSpectrum(self.my_spectrum.value)
        # operate on the spectrum object...
```

10.4.7 SpecWavelength

```
class SpecWavelength(SpecFloat):
    def __init__(self, label, datacube, defaultValue=None):
```

A SpecWavelength accepts a datacube as an argument and allows the user to select a wavelength that is present in that cube. It's uder interface is a slider with range equal to the wavelength range in the passed in datacube.

Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecWavelength

class MyPlugin(CubePlugin):
```

```
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        self.my_wavelength = SpecWavelength(label='Wavelength',
                                            datacube=self.datacube,
                                            defaultValue=None)
        self.my_wavelength.help = 'my mouse hover help'

    def action(self):
       # a float
       wavelength = self.my_wavelength.value

       # a band at a user selected wavelength
       band = self.datacube.getBandAtWavelength(self.my_wavelength.value)
```

10.4.8 SpecBandNumber

```
class SpecBandNumber(SpecInt):
    def __init__(self, label, datacube, defaultValue=0):
```

A SpecBandNumber allows the user to select one of the bands present in a datacube by index. Example:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecBandNumber

class MyPlugin(CubePlugin):
    """
    An example plugin
    """
    label = "My Plugin"
    userLevel = 1

    def setup(self):
        self.my_band_num = SpecBandNumber(label='Band',
                                          datacube=self.datacube,
                                          defaultValue=0)
        self.my_band_num.help = 'my mouse hover help'

    def action(self):
       band = self.datacube.getBand(self.my_band_num.value)
```

10.5 Using the Workbench

Plugins automatically receive a reference to the Spectronon workbench, `self.wb`. The workbench gives you a lot of freedom (including the freedom to break things!), so use this reference wisely. Most often, you will use the workbench to get references to other datacubes or spectra on the resource tree, add additional items to the resource tree, or prompt the user for a filename or other information. SourceIDs in the below methods refer to IDs returned from SpecCubes or SpecSpectra.

Some important methods you can call are:

```
def getCube(self, sourceID=None):
    """
    Returns the requested datacube.  If sourceID is None,
    returns the current cube.
    """

def getRendering(self, renderID=None):
    """
    Returns the requested rendering.  If sourceID is None,
    returns the current rendering.
    """

def getSpectrum(self, sourceID=None):
    """
    Returns the requested spectrum.  If sourceID is None,
    returns the current spectrum.
    """

def addCube(self, datacube, name=None, render=True):
    """
    Adds a datacube to the workbench.
    """

def addSpectrum(self, spectrumObject, name=None):
    """
    Adds a spectrum to the workbench.
    """

def postMessage(self, message, title=''):
    """
    Post a message in a dialog box
    """

def postScrolledMessage(self, message, title=''):
    """
    Post a message in a dialog box that can be scrolled and text
    can be selected.  Good for long messages
    """

def postQuestion(self, message, title="Proceed?"):
    """
    post message.  returns True if user selects 'OK' and False if 'Cancel'

    params:
        message: the message to show
        title: the dialog box title
    """

def requestOpenFilename(self,
                        wildcard='',
                        message="Enter Filename To Open ",
                        multiple=False):
    """
    Requests a filename(s) from the user.  Returns
    the filename as a string, or a list of strings
    if multiple=True.  Returns None if the user selects
    'Cancel'.
    """
```

```
def requestDirectory(self,
                     message="Please give a directory path",
                     suggest=""):
    """
    Requests a directory form the user.  Returns
    the selected directory path as a string.  Returns
    None if the user selects 'Cancel'.
    """
```

10.6 Using Datacubes

Plugins get a reference to their relevant datacube as `self.datacube`. It is possible to get references to other datacubes directly from the workbench. Datacubes contain a data array in numpy format as well as a collection of metadata.

Some important methods you can call are:

```
def getSpectrumArray(self, sample, line, asFloat=False):
    """
    return a 1D array at location (sample, line)

    params
        sample: the sample number (int)
        line: the line number (int)
        asFloat: scale the datacube from 0-1 (bool)

    returns
        1D array of length 'bands'
    """

def getBandNumForWavelength(self, wavelength):
    """
    return the band number nearest the given wavelength

    params
        wavelength: wavelength of light in nanometers(float)
    returns
     (2d array)
    """

def getBandCount(self):
    """return The number of bands in this cube
    (aka. the number of different wavelengths)
    """

def getSampleCount(self):
    """
    return the number of samples for the cube
    (aka. the image height)
    """

def getLineCount(self):
    """
    return the number of lines for the cube
    (aka. the image width)
    """
```

```
def getBandAtWavelength(self, wavelength, asFloat=False):
    """
    return band at given wavelength

    params
        wavelength: the wavelength of the band you want (float)
        asFloat: scale the datacube from 0-1 (bool)
    returns
        the band for the given wavelength frequency (2d array)

    return the band with the closest wavelength to the given value
    in the case of equidistance between two bands, the lower of the two
    bands is returned
    """

def getBandWithName(self, name, asFloat=False):
    """
    return band with given name

    params
        name: the name of the band you want (float)
        asFloat: scale the datacube from 0-1 (bool)
    returns
        the band with the given name  (2d array)

    you may pass the string form of a known wavelength
    """

 def getBand(self, bandnumber, asFloat=False):
    """
    Return band at given band number

    params
        bandnumber: band number between 0 and bands - 1 (integer)
        asFloat: scale the datacube from 0-1 (bool)
    returns
        band: the image at the band number (2d array) always (lines, samples)
    """

  def getFrame(self, line, asFloat=False, asBIP=False):
    """
    return frame at given line number

    params
        line: the line number (first = 0) of a frame in the cube (int)
        asFloat: scale the datacube from 0-1 (bool)
    returns:
        frame: the spectrum of each pixel in a line of the image (2D array)
        always (samples, bands)
    """

def appendFrame(self, frame):
    """
    append frame to cube

    params
        frame: the content of a frame of the same (samples, bands)
            dimensions as the cube (2d array)
```

```
    appends a frame and increases number of lines in the cube by one
    """

def appendBandWithName(self, band, bandname=None):
    """
    append band to cube

    params
        band: the content of a band of the same (lines, samples)
            dimensions as the cube (2d array)

    appends a band and increases number of bands in the cube by one
    """

def appendBandWithWavelength(self, band, wave=None):
    """
    append band to cube

    params
        band: the content of a band of the same (lines, samples)
            dimensions as the cube (2d array)
    keywords
        writeheader: default (True).  if headerfile exists, save header
            file to disk to reflect new larger dimensions of cube.

    appends a band and increases number of bands in the cube by one
    """

def getSubCube(self, minsample=0, minline=0, maxsample= -1,
               maxline= -1, mode="memory", interleave=None):
    """getSubCube

        return a cube with the data in the given range, mode, and interleave

        with no arguments, this will return a complete copy.
    """

def getFramelessCopy(self, makeTypeFloat=False, mode=None, asBIP=False):
    """
    return an empty cube object with a header based on the header of this cube

    optional makeTypeFloat will return
    """

def getBandlessCopy(self, makeTypeFloat=False, mode=None):
    """
    return an empty cube object with a header based on the header of this cube

    optional makeTypeFloat will return
    """
    newcube = self._getEmptyCubeWithHeaderCopy(makeTypeFloat, mode)
    newcube.setMetaValue("bands", 0)
    return newcube

def setBand(self, bandnumber, band):
    """
    writes given band at given band
```

```
    params
        bandnumber: the number of the band to overwrite
    returns
        None
    """

def setFrame(self, line, frame):
    """
    write a given frame into location at given line

    params
        line: the line number to overwrite (int)
        frame: the content of the frame (2d array)
    returns
        None

    This is only for replacing frames that are already there.
    to add Frames to a cube use appendFrame
    """

 def getArray(self, asFloat=False, asBIP=False):
     """
     get the numpy data array for this datacube

     params:
        asFloat: scale the datacube from 0-1 (bool)
        asBIP: force the data to be in BIP interleave (lines, samples, bands)
     """
```

Additionally, the function `util.makeEmptyCube()` is convenient for constructing new datacubes. It's usage is as follows:

```
from resonon.core.data import util

newcube = util.makeEmptyCube(mode="memory",
                             typechar='f',
                             rotationString=datacube.getRotationString())
```

10.7 Example Plugins

The following plugin examples are based on built-in plugins from Spectronon.

> **Warning:** Each plugin in Spectronon needs to have a distinct class name. If you change the name of the following plugins, don't just remove 'Example' from the class name or your plugin will conflict with a built-in plugin.

10.7.1 Cube Plugins

Cube plugins are the most common. They operate on a datacube and return a datacube. Cube plugins add themselves to the *New Cube* submenu and context menu.

A very simple example based on a Spectronon built-in plugin is shown below:

```
from spectronon.workbench.plugin import CubePlugin
from resonon.utils.spec import SpecWavelength
```

```
from resonon.core.data import util

class BandRatioExample(CubePlugin):
    """Create a new single band cube that results from divide two bands of an
       existing cube"""  # The plugin's help text

    label = "Band Ratio Example"  #name of plugin as it appears in Spectronon

    # default rendering, others include SingleBand, ThreshToColor, TriBand, etc
    defaultRenderer = "SingleBand"

    # allows control for what  which users the plugin is available.
    # 1 is for normal users
    userLevel = 1

    def setup(self):
        # GUI controls that allows the user to select wavelengths, as they
        # exist in 'datacube'
        self.numband  = SpecWavelength(label="Numerator Wavelength",
                                       datacube = self.datacube)
        self.denomband  = SpecWavelength(label="Denominator Wavelength",
                                         datacube = self.datacube)

    def action(self):
        #make an empty cube for the results, preserving metadata from the old cube
        newcube = util.makeEmptyCube(mode="memory",
                                     typechar='f',
                                     rotationString=self.datacube.getRotationString())

        #get the data array from the datacube at the two specific wavelengths
        topband = self.datacube.getBandAtWavelength(self.numband.value,
                                                    asFloat=True)
        bottomband =self.datacube.getBandAtWavelength(self.denomband.value,
                                                      asFloat=True)

        #perform the math on the numpy arrays
        result = topband / bottomband

        #put the results in the empty cube
        newcube.appendBandWithName(result,bandname=self.label)

        #return the new cube and it will be added to the Spectronon file tree
        return newcube
```

10.7.2 Render Plugins (Image Plugins)

Render Plugins create a 2D representation of a datacube, known as an 'Image' in Spectronon. They operate on a cube and return a 2D image of 3 or fewer bands ('RGB', or 'colors'). A simple example is shown below:

```
import numpy
from spectronon.workbench.plugin import RenderPlugin

class BandAverageExample(RenderPlugin):
    "returns a greyscale image of the average across all bands"
    label = "Band Average Example"
    userLevel = 1
```

```
    def action(self):

        datacube = self.datacube

        #create an numpy array of the same spatial dimensions of the datacube
        out = numpy.zeros((datacube.getLineCount(), datacube.getSampleCount()),
                          dtype=numpy.float32)
        for band in range(datacube.getBandCount()):
            #for each band, add it to the accumulator
            out += datacube.getBand(band)

        #normalize the results by the number of bands
        out = out/datacube.getBandCount()

        return out
```

10.7.3 Filter Plugins

Filter Plugins operate on a render to enhance or otherwise filter its appearance. It is currently not possible for a user to write a Filter Plugin.

10.7.4 Select Plugins

Select Plugins operate on a Region of Interest created in Spectronon with the Lasso, Wand, or Marquee tools. An example is shown below that finds the spectral and spatial average value of the pixels inside of the ROI.

```
from spectronon.workbench.plugin import SelectPlugin

class AverageROI_Example(SelectPlugin):
    '''
    Calculate a single mean across all bands in a selected region
    '''
    label = "ROI Example"
    userLevel = 1

    def action(self):
        # get the raw numpy data from the datacube
        # a BIP is indexed as (lines, samples, bands)
        cube_array = self.datacube.getArray(asBIP=True)

        # Use a numpy slice to get the portion of the array that has been selected
        # the pointlist is given in order(sample, line).
        subarray = cube_array[self.pointlist[:, 1], self.pointlist[:, 0], :]

        # calculate the average of the selected region
        ave = subarray.mean()

        # display the calculated result for the user
        self.wb.postMessage('ROI Average : %s' % ave, title="Average")
```

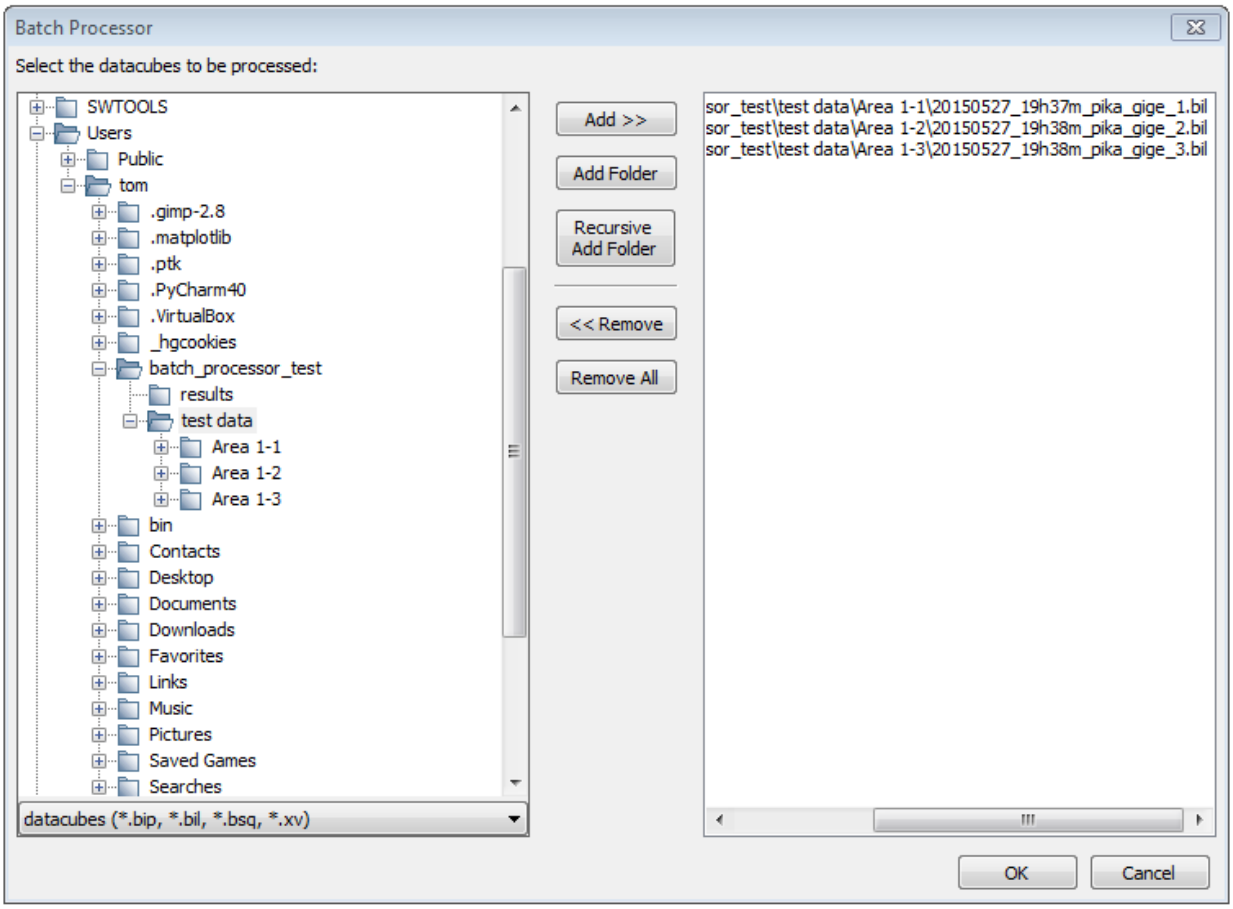
CHAPTER

ELEVEN

BATCH PROCESSING

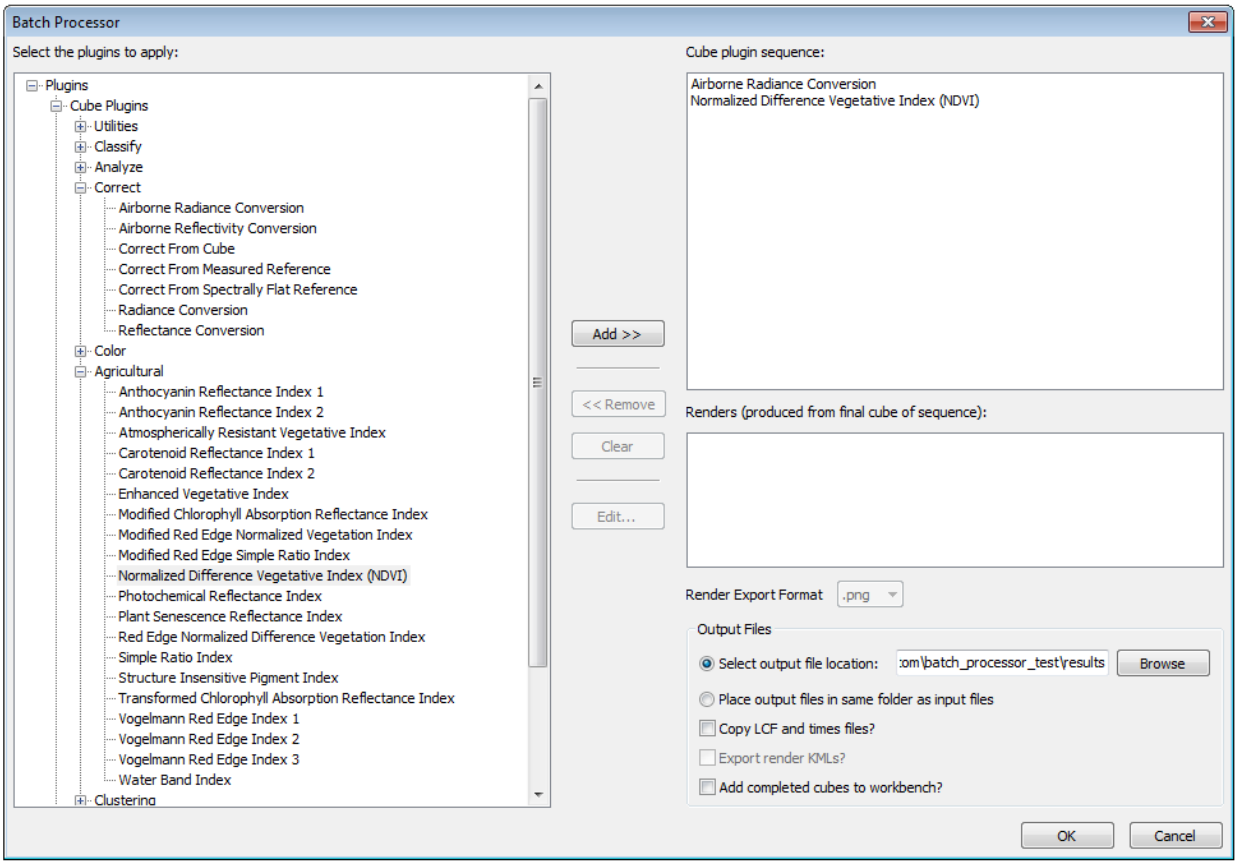
Manually running the same processing sequence on a series of datacubes can be a tedious process. Spectronon's batch processor allows the user to choose a number of datacubes at once and set up a sequence of processing steps (plugins) to execute on the chosen files. This may be of particular interest to users of our Airborne product, but applies to any customer who needs process multiple datacubes at once.

To start the batch processor, select **File** → **Batch Processor.** The batch processing window appears.



Expand the directory tree on the left side of the window to find the datacubes you wish to process. Use the buttons in the middle of the screen to select the desired datacubes. Datacubes that will be processed appear on the right side of the window. In the figure above 3 datacubes are selected for processing.

When you are happy with your selection, click **Ok**. A new window appears to define the plugins you wish to run.

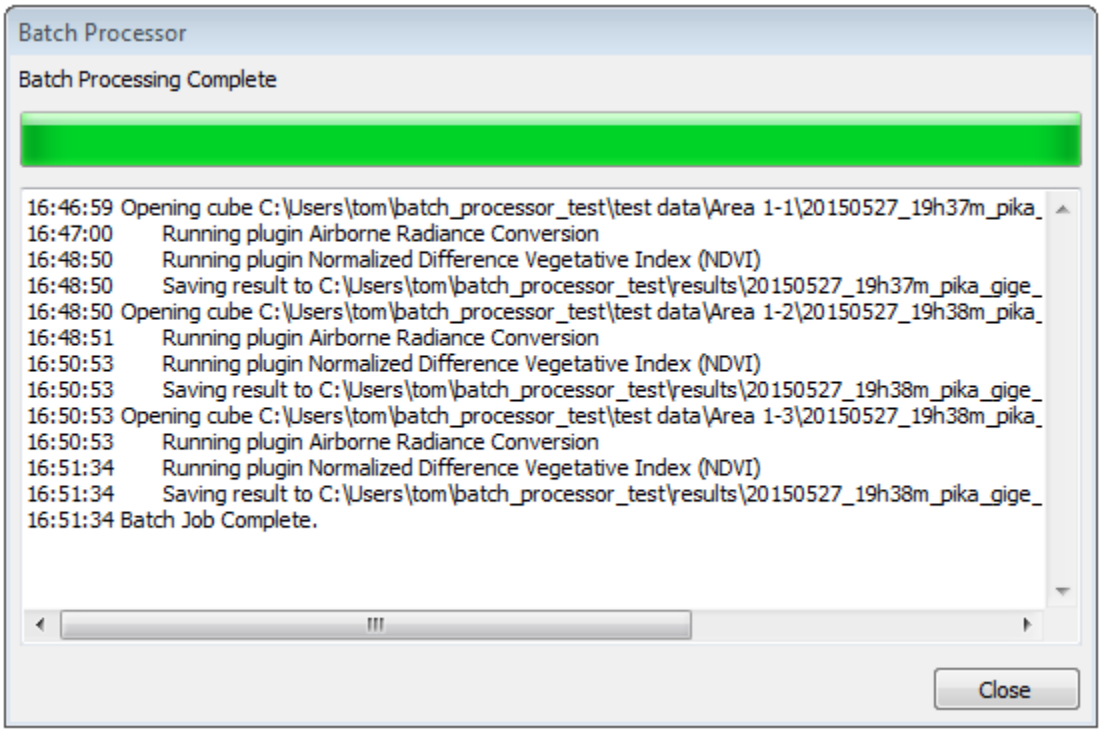


Use the directory tree to select the cube plugins you wish to run. Cube plugins generate new datacubes and will run in the order selected, as shown in the **Cube plugin sequence** panel. Each plugin operates on the output of the plugin before it. Each plugin's setup window appears as you add it to the processing queue. In the figure above the "Airborne Radiance Conversion" plugin is set to run, followed by the NDVI plugin.

Select any render plugins you wish to run. Render plugins generate images from the final result of the cube plugin sequence, and appear in the **Renders** panel. You can choose to save renders as .png, .tiff or .jpg with **Render Export Format**. In the figure above no render plugins are selected, so no images will be generated.

Use the controls at the lower right to select the location to save results. **Copy LCF and times files?** and **Export render KMLs?** are useful for our airborne customers. If desired, the resultant datacubes can be added to the workbench to show in Specrtonon when processing is finished.

Press the **OK** button to begin processing. A progress dialog appears.


Batch Processor
Batch Processing Complete
16:46:59 Opening cube C:\Users\tom\batch_processor_test\test data\Area 1-1\20150527_19h37m_pika_
16:47:00 Running plugin Airborne Radiance Conversion
16:48:50 Running plugin Normalized Difference Vegetative Index (NDVI)
16:48:50 Saving result to C:\Users\tom\batch_processor_test\results\20150527_19h37m_pika_gige_
16:48:50 Opening cube C:\Users\tom\batch_processor_test\test data\Area 1-2\20150527_19h38m_pika_
16:48:51 Running plugin Airborne Radiance Conversion
16:50:53 Running plugin Normalized Difference Vegetative Index (NDVI)
16:50:53 Saving result to C:\Users\tom\batch_processor_test\results\20150527_19h38m_pika_gige_
16:50:53 Opening cube C:\Users\tom\batch_processor_test\test data\Area 1-3\20150527_19h38m_pika_
16:50:53 Running plugin Airborne Radiance Conversion
16:51:34 Running plugin Normalized Difference Vegetative Index (NDVI)
16:51:34 Saving result to C:\Users\tom\batch_processor_test\results\20150527_19h38m_pika_gige_
16:51:34 Batch Job Complete.
Close

Press the **Close** button to dismiss the dialog after processing is complete. The results are saved to the selected location.

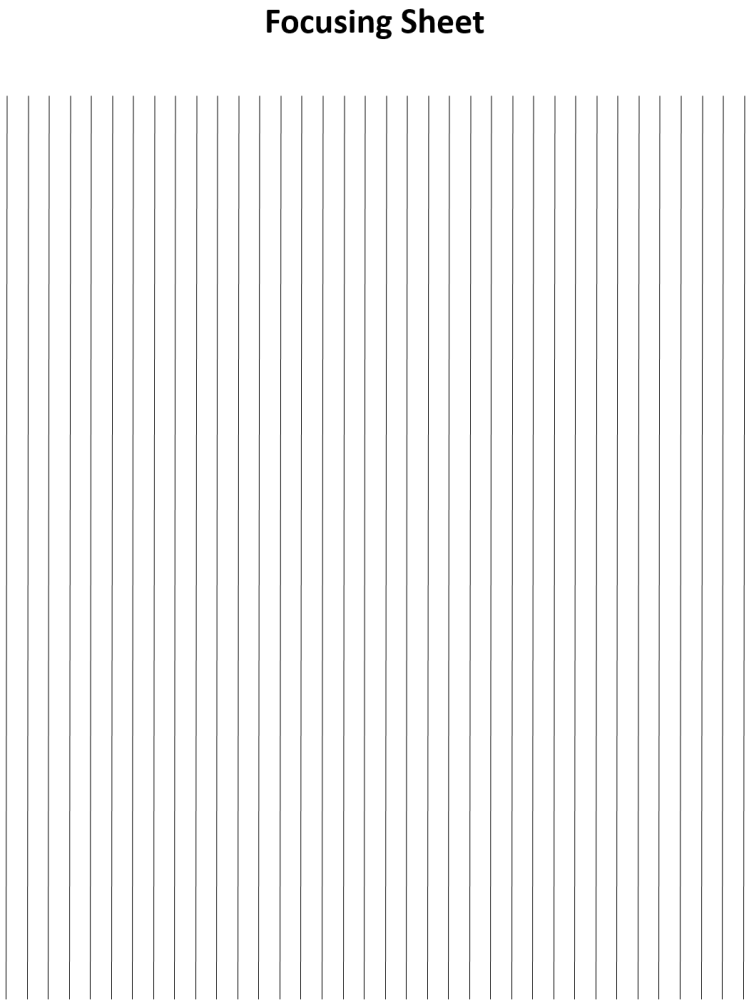
CHAPTER

TWELVE

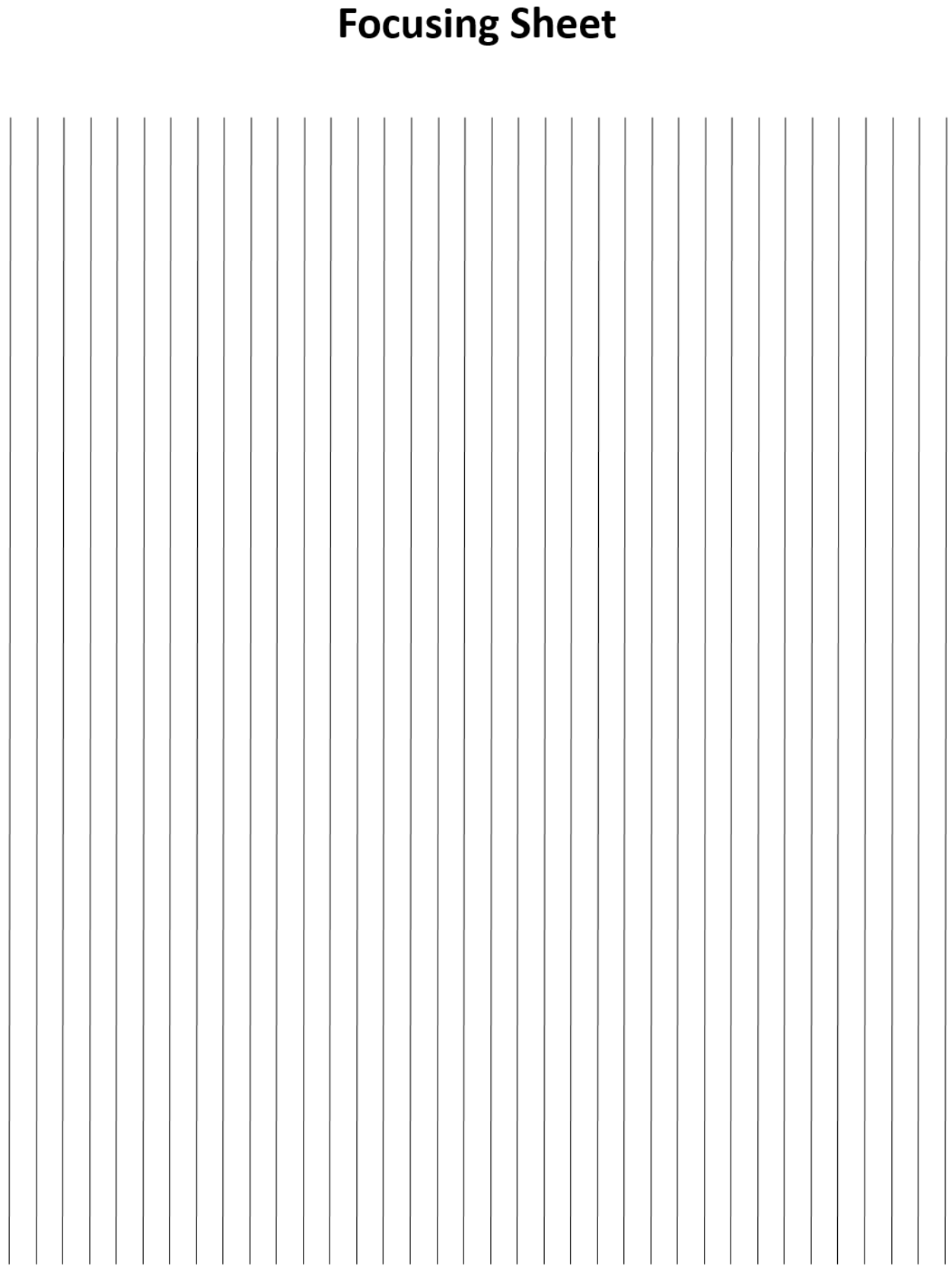
FOCUSING & CALIBRATION SHEETS

Use the focusing sheets to focus the objective lens, and use the aspect ratio calibration sheet to set the stage speed and imager framerate. See the chapter on Basic Data Acquisition for details.

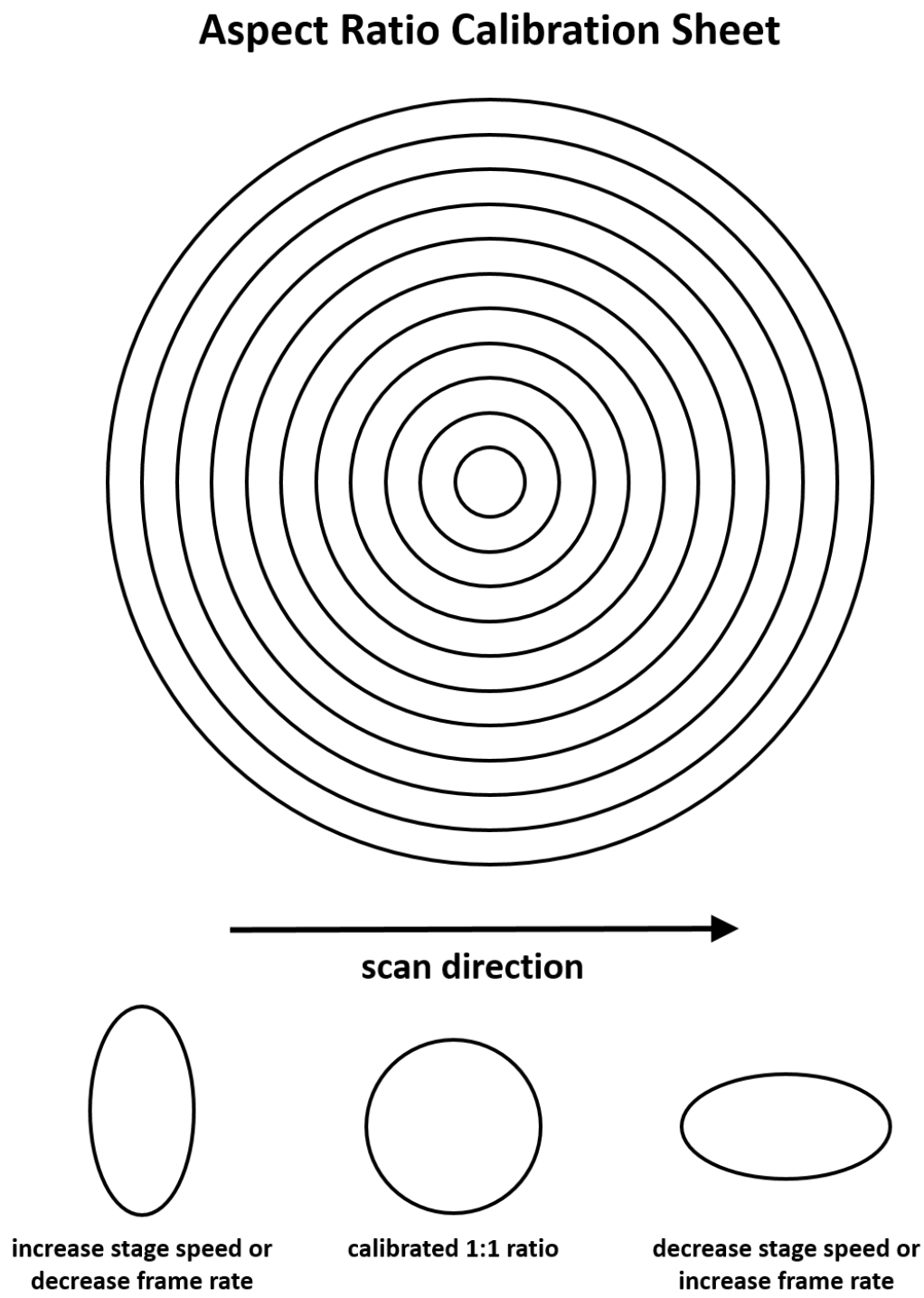
12.1 Small Focusing Sheet



12.2 Large Focusing Sheet



12.3 Aspect Ratio Calibration Sheet


Aspect Ratio Calibration Sheet
scan direction
increase stage speed or decrease frame rate
calibrated 1:1 ratio
decrease stage speed or increase frame rate

CHAPTER

THIRTEEN

CONTACT US FOR PRODUCT SUPPORT

Phone: (406) 586-3356
Website: http://www.resonon.com
Email: productsupport@resonon.com
Also see our product support webpage located at:
http://www.resonon.com/customer_support.html

CHAPTER

FOURTEEN

COPYRIGHT NOTICE

Copyright © March 2014 Resonon Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means (electronic or mechanical, including photocopying) for any purpose without written permission from Resonon. Resonon will not be responsible or liable for any accidental or inevitable damage that may result from unauthorized access or modifications.

This document may contain errors or inaccuracies, and it may be revised without advance notice. This manual is updated frequently.

Please see our software copyright and EULA at
http://www.spectronon.com/docs/Spectronon_EULA.pdf

Resonon welcomes any recommendations regarding this manual. Customer feedback is always welcome as it helps us to continuously improve upon the quality of our products.

Please contact us at Resonon with any questions or comments. We look forward to hearing from you.